Part III — Quantum Computation

Based on lectures by R. Jozsa

Notes taken by Dexter Chua

Michaelmas 2016

These notes are not endorsed by the lecturers, and I have modified them (often significantly) after lectures. They are nowhere near accurate representations of what was actually lectured, and in particular, all errors are almost surely mine.

Quantum mechanical processes can be exploited to provide new modes of information processing that are beyond the capabilities of any classical computer. This leads to remarkable new kinds of algorithms (so-called quantum algorithms) that can offer a dramatically increased efficiency for the execution of some computational tasks. Notable examples include integer factorisation (and consequent efficient breaking of commonly used public key crypto systems) and database searching. In addition to such potential practical benefits, the study of quantum computation has great theoretical interest, combining concepts from computational complexity theory and quantum physics to provide striking fundamental insights into the nature of both disciplines.

The course will cover the following topics:

Notion of qubits, quantum logic gates, circuit model of quantum computation. Basic notions of quantum computational complexity, oracles, query complexity.

The quantum Fourier transform. Exposition of fundamental quantum algorithms including the Deutsch-Jozsa algorithm, Shor's factoring algorithm, Grovers searching algorithm.

A selection from the following further topics (and possibly others):

(i) Quantum teleportation and the measurement-based model of quantum computation;

(ii) Lower bounds on quantum query complexity;

(iii) Phase estimation and applications in quantum algorithms;

(iv) Quantum simulation for local hamiltonians.

Pre-requisites

It is desirable to have familiarity with the basic formalism of quantum mechanics especially in the simple context of finite dimensional state spaces (state vectors, Dirac notation, composite systems, unitary matrices, Born rule for quantum measurements). Prerequisite notes will be provided on the course webpage giving an account of the necessary material including exercises on the use of notations and relevant calculational techniques of linear algebra. It would be desirable for you to look through this material

at (or slightly before) the start of the course. Any encounter with basic ideas of classical theoretical computer science (complexity theory) would be helpful but is not essential.

Contents

0 Introduction **4**

1 Classical computation theory **5**

2 Quantum computation **9**

3 Some quantum algorithms **13**
3.1 Balanced vs constant problem 13
3.2 Quantum Fourier transform and periodicities 15
3.3 Shor's algorithm 19
3.4 Search problems and Grover's algorithm 25
3.5 Amplitude amplification 31

4 Measurement-based quantum computing **34**

5 Phase estimation algorithm **41**

6 Hamiltonian simulation **44**

Index **50**

0 Introduction

Quantum computation is currently a highly significant and important subject, and is very active in international research.

First of all, it is a fundamental connection between physics and computing. We can think of physics as computing, where in physics, we label states with parameters (i.e. numbers), and physical evolution changes these parameters. So we can think of these parameters as encoding information, and physical evolution changes the information. Thus, this evolution can be thought of as a computational process.

More strikingly, we can also view computing as physics! We all have computers, and usually represent information as bits, 0 or 1. We often think of computation as manipulation of these bits, i.e. as discrete maths. However, there is no actual discrete bits — when we build a computer, we need physical devices to represent these bits. When we run a computation on a computer, it has to obey the laws of physics. So we arrive at the idea that the limits of computation are not a part of mathematics, but depend on the laws of physics. Thus, we can associate a "computing power" with any theory of physics!

On the other hand, there is also a technology/engineering aspect of quantum computation. Historically, we have been trying to reduce the size of computers. Eventually, we will want to try to achieve miniaturization of computer components to essentially the subatomic scale. The usual boolean operations we base our computations on do not work so well on this small scale, since quantum effects start to kick in. We could try to mitigate these quantum issues and somehow force the bits to act classically, but we can also embrace the quantum effects, and build a quantum computer! There is a lot of recent progress in quantum technology. We are now expecting a 50-qubit quantum computer in full coherent control soon. However, we are not going to talk about implementation in this course.

Finally, apart from the practical problem of building quantum computers, we also have theoretical quantum computer science, where we try to understand how quantum algorithms behave. This is about how we can actually exploit quantum physical facts for computational possibilities beyond classical computers. This will be the focus of the course.

1 Classical computation theory

To appreciate the difference between quantum and classical computing, we need to first understand classical computing. We will only briefly go over the main ideas instead of working out every single technical detail. Hence some of the definitions might be slightly vague.

We start with the notion of "computable". To define computability, one has to come up with a sensible mathematical model of a computer, and then "computable" means that theoretical computer can compute it. So far, any two sensible mathematical models of computations we manage to come up with are equivalent, so we can just pick any one of them. Consequently, we will not spend much time working out a technical definition of computable.

Example. Let N be an integer. We want to figure out if N a prime. This is clearly computable, since we can try all numbers less than N and see if it divides N.

This is not too surprising, but it turns out there are some problems that are not computable! Most famously, we have the Halting problem.

Example (Halting problem)**.** Given the code of a computer program, we want to figure out if the computer will eventually halt. In 1936, Turing proved that this problem is uncomputable! So we cannot have a program that determines if an arbitrary program halts.

For a less arbitrary problem, we have

Example. Given a polynomial with integer coefficients with many variables, e.g. $2x^2y - 17zw^{19} + x^5w^3 + 1$, does this have a root in the integers? It was shown in 1976 that this problem is uncomputable as well!

These results are all for classical computing. If we expect quantum computing to be somehow different, can we get around this problems? This turns out not to be the case, for the very reason that all the laws of quantum physics (e.g. state descriptions, evolution equations) are all computable on a classical computer (in principle). So it follows that quantum computing, being a quantum process, cannot compute any classical uncomputable problem.

Despite this limitation, quantum computation is still interesting! In practice, we do not only care about computability. We care about how efficient we are at doing the computation. This is the problem of complexity — the complexity of a quantum computation might be much simpler than the classical counterpart.

To make sense of complexity, we need to make our notion of computations a bit more precise.

Definition (Input string)**.** An *input bit string* is a sequence of bits $x = i_1 i_2 \cdots i_n$, where each i_k is either 0 or 1. We write B_n for the set of all n-bit string, and $B = \bigcup_{n\in\mathbb{N}} B_n$. The *input size* is the length n. So in particular, if the input is regarded as an actual number, the size is not the number itself, but its logarithm.

Definition (Language)**.** A *language* is a subset $L \subseteq B$.

Definition (Decision problem)**.** Given a language L, the *decision problem* is to determine whether an arbitrary $x \in B$ is a member of L. The output is thus 1 bit of information, namely yes or no.

Of course, we can have a more general task with multiple outputs, but for simplicity, we will not consider that case here.

Example. If L is the set of all prime numbers, then the corresponding decision problem is determining whether a number is prime.

We also have to talk about models of computations. We will only give an intuitive and classical description of it.

Definition (Computational model)**.** A *computational model* is a process with discrete steps (elementary computational steps), where each step requires a constant amount of effort/resources to implement.

If we think about actual computers that works with bits, we can imagine a step as an operation such as "and" or "or". Note that addition and multiplication are *not* considered a single step — as the number gets larger, it takes more effort to add or multiply them.

Sometimes it is helpful to allow some randomness.

Definition (Randomized/probabilistic computation)**.** This is the same as a usual computational model, but the process also has access to a string $r_1, r_2, r_3, \cdots$ of independent, uniform random bits. In this case, we will often require the answer/output to be correct with "suitably good" probability.

In computer science, there is a separate notion of "non-deterministic" computation, which is *different* from probabilistic computation. In probabilistic computation, every time we ask for a random number, we just pick one of the possible output and follows that. With a non-deterministic computer, we simultaneously consider *all* possible choices with no extra overhead. This is extremely powerful, and also obviously physically impossible, but it is a convenient thing to consider theoretically.

Definition (Complexity of a computational task (or an algorithm))**.** The *complexity* of a computational task or algorithm is the "consumption of resources as a function of input size n". The resources are usually the time

$$T(n) = \text{number of computational steps needed},$$

and space

$$Sp(n) = \text{number of memory/work space needed}.$$

In each case, we take the worse case input of a given size n.

We usually consider the worst-case scenario, since, e.g. for primality testing, there are always some numbers which we can easily rule out as being not prime (e.g. even numbers). Sometimes, we will also want to study the average complexity.

In the course, we will mostly focus on the time complexity, and not work with the space complexity itself.

As one would imagine, the actual time or space taken would vary a lot on the actual computational model. Thus, the main question we ask will be whether $T(n)$ grows polynomially or super-polynomially ("exponentially") with n.

Definition (Polynomial growth)**.** We say $T(n)$ *grows polynomially*, and write

$$T(n) = O(\text{poly}(n)) = O(n^k)$$

for some k, if there is some constant c, and some integer k and some integer n_0 such that $T(n) < cn^k$ for all $n > n_0$.

The other possible cases are exponential growth, e.g. $T(n) = c_1 2^{c_2 n}$, or super-polynomial and sub-exponential growth such as $T(n) = 2^{\sqrt{n}}$ or $n^{\log n}$.

We will usually regard polynomial time processes as "feasible in practice", while super-polynomial ones are considered "infeasible". Of course, this is not always actually true. For example, we might have a polynomial time of $n^{10^{10^{10}}}$, or an exponential time of $2^{0.0000\ldots0001n}$. However, this distinction of polynomial vs non-polynomial is robust, since any computational model can "simulate" other computational models in polynomial time. So if something is polynomial in one computational model, it is polynomial in all models.

In general, we can have a more refined complexity classes of decision problems:

(i) ***P*** (*polynomial time*): The class of decision problems having *deterministic* polynomial-time algorithm.

(ii) ***BPP*** (*bounded error, probabilistic polynomial time*): The class of decision problems having *probabilistic* polynomial time algorithms such that for every input,

$$\text{Prob(answer is correct)} \geq \frac{2}{3}.$$

The number $\frac{2}{3}$ is sort of arbitrary — we see that we cannot put $\frac{1}{2}$, or else we can just randomly guess a number. So we need something greater than $\frac{1}{2}$, and "bounded" refers to it being bounded away from $\frac{1}{2}$. We could replace $\frac{2}{3}$ with any other constant $\frac{1}{2} + \delta$ with $0 < \delta < \frac{1}{2}$, and **BPP** is the same. This is because if we have a $\frac{1}{2} + \delta$ algorithm, we simply repeat the algorithm K times, and take the majority vote. By the Chernoff bound (a result in probability), the probability that the majority vote is correct is $> 1 - e^{-2\delta^2 K}$. So as we do more and more runs, the probability of getting a right answer grows exponentially. This can be bigger than an $1 - \varepsilon$ by a suitably large K. Since K times a polynomial time is still polynomial time, we still have a polynomial time algorithm.

These two are often considered as "classically feasible computations", or "computable in practice". In the second case, we tolerate small errors, but that is fine in practice, since in genuine computers, random cosmic rays and memory failures can also cause small errors in the result, even for a deterministic algorithm.

It is clear that **P** is contained in **BPP**, but we do not know about the other direction. It is not known whether **P** and **BPP** are the same — in general, not much is known about whether two complexity classes are the same.

Example (Primality testing)**.** Let N be an integer. We want to determine if it is prime. The input size is $\log_2 N$. The naive method of primality testing is to test all numbers and see if it divides N. We only need to test up to $\sqrt{N}$, since if N has a factor, there must be one below $\sqrt{N}$. The is *not* polynomial time, since we need $\sqrt{N} = 2^{\frac{1}{2}\log N}$ operations, we see that this is exponential time.

How about a probabilistic algorithm? We can choose a random $k < N$, and see if k divides N. This is a probabilistic, polynomial time algorithm, but it is not bounded, since the probability of getting a correct answer is not $> \frac{1}{2}$.

In reality, primality testing is known to be in **BPP** (1976), and it is also known to be in **P** (2004).

Finally, we quickly describe a simple model of (classical) computation that we will use to build upon later on. While the most famous model for computation is probably the *Turing machine*, for our purposes, it is much simpler to work with the *circuit model*.

The idea is simple. In general, we are working with bits, and a program is a function $f : B_m \to B_n$. It is a mathematical fact that any such function can be constructed by combinations of boolean AND, OR and NOT gates. We say that this is a *universal set* of gates. Thus a "program" is a specification of how to arrange these gates in order to give the function we want, and the time taken by the circuit is simply the number of gates we need.

Of course, we could have chosen a different universal set of gates, and the programs would be different. However, since only a fixed number of gates is needed to construct AND, OR and NOT from any universal set, and vice versa, it follows that the difference in time is always just polynomial.

2 Quantum computation

We are now going to start talking about quantum computation. Our model of quantum computation will be similar to the circuit model.

The main difference is that instead of working with bits, we work with *qubits*. A single qubit is an element of $\mathbb{C}^2$, with basis vectors

$$|0\rangle = \begin{pmatrix}1\\0\end{pmatrix}, \quad |1\rangle = \begin{pmatrix}0\\1\end{pmatrix}.$$

When we have multiple qubits, we write them as $|a\rangle\,|b\rangle$, which is a shorthand for $|a\rangle \otimes |b\rangle$ etc.

Now any classical bit string $x = i_1 i_2 \cdots i_n$ can be encoded as a qubit

$$|i_1\rangle\,|i_2\rangle \cdots |i_n\rangle\,|0\rangle \cdots |0\rangle \in \bigotimes_{i=0}^{n+k} \mathbb{C}^2 \cong \mathbb{C}^{2^{n+k}},$$

where we padded k extra zeroes at the end. In classical computation, there was no such need, because within any computation, we were free to introduce or remove extra bits. However, one peculiarity of quantum computation is that all processes are invertible, and in particular, the number of qubits is always fixed. So if we want to do something "on the side" during the computations, the extra bits needed to do so must be supplied at the beginning.

Now the quantum gates are not just boolean functions, but *unitary operators* on the states. The standard gates will operate on one or two qubits only, and we can chain them together to get larger operators.

We now list our standard unitary gates. The four main (families) single-qubit gates we will need are the following (in the standard $|0\rangle, |1\rangle$ basis):

$$\mathsf{X} = \begin{pmatrix}0 & 1\\1 & 0\end{pmatrix}, \quad \mathsf{Z} = \begin{pmatrix}1 & 0\\0 & -1\end{pmatrix}, \quad \mathsf{H} = \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\1 & -1\end{pmatrix}, \quad \mathsf{P}_\varphi = \begin{pmatrix}1 & 0\\0 & e^{\varphi}\end{pmatrix}$$

We also have two "controlled" gates. These controlled gates take in two qubits. It does not directly change the first qubit, and will decide whether or not to act on the second bit depending on the value of the first. They are given by

$$\mathsf{CX}\,|i\rangle\,|j\rangle = |i\rangle\,X^i\,|j\rangle, \quad \mathsf{CZ}\,|i\rangle\,|j\rangle = |i\rangle$$

In the basis $\{|0\rangle\,|0\rangle, |0\rangle\,|1\rangle, |1\rangle\,|0\rangle, |1\rangle\,|1\rangle\}$, we can write these operators as

$$\mathsf{CX} = \begin{pmatrix}1 & 0 & 0 & 0\\0 & 1 & 0 & 0\\0 & 0 & 0 & 1\\0 & 0 & 1 & 0\end{pmatrix}, \quad \mathsf{CZ} = \begin{pmatrix}1 & 0 & 0 & 0\\0 & 1 & 0 & 0\\0 & 0 & 1 & 0\\0 & 0 & 0 & -1\end{pmatrix},$$

These will be taken as the basic unitary gates.

The other thing we are allowed to do in a quantum system is *measurement*. What does this do? We wlog we are measuring the first qubit. Suppose the state before the measurement is given by

$$c_0\,|0\rangle\,|a\rangle + c_1\,|1\rangle\,|b\rangle,$$

where $|a\rangle, |b\rangle$ are $(n-1)$-qubit states of unit norm, and $|c_0|^2 + |c_1|^2 = 1$.

Then when we measure the first qubit, we have probability $|c_0|^2$ of getting 0, and probability $|c_1|^2$ of getting 1. After measuring, the resulting state is either $|0\rangle |a\rangle$ or $|1\rangle |b\rangle$, depending on whether we measured 0 or 1 respectively.

Measurements are irreversible, and in particular aren't given by unitary matrices. We will allow ourselves to make classical computations based on the results of the measurements, and decide which future quantum gates we apply based on the results of the classical computations.

While this seems like a very useful thing to do, it is a mathematical fact that we can modify such a system to an equivalent quantum one where all the measurements are done at the end instead. So we are not actually adding anything new to our mathematical model of computation by allowing such classical manipluations.

Now what is the analogous notion of a universal set of gates? In the classical case, the set of boolean functions is discrete, and therefore a finite set of gates can be universal. However, in the quantum case, the possible unitary matrices are continuous, so no finite set can be universal (more mathematically, there is an uncountable number of unitary matrices, but a finite collection of gates can only generate a countable subgroup of matrices).

Thus, instead of asking for universality, we ask for approximate universality. To appreciate this, we can take the example of rotations — there is no single rotation that generates all possible rotations in $\mathbb{R}^2$. However, we can pick a rotation by an irrational angle, and then the set of rotations generated by this rotation is dense in the set of all rotations, and this is good enough.

Definition (Approximate universality)**.** A collection of gates is *approximately universal* if for any unitary matrix U and any $\varepsilon > 0$, there is some circuit $\tilde{U}$ built out of the collection of gates such that

$$\left\| U - \tilde{U} \right\| < \varepsilon.$$

In other words, we have

$$\sup_{\|\psi\|=1} \left\| U|\psi\rangle - \tilde{U}|\psi\rangle \right\| < \varepsilon,$$

where we take the usual norm on the vectors (any two norms are equivalent if the state space is finite dimensional, so it doesn't really matter).

We will provide some examples without proof.

Example. The infinite set $\{\mathsf{CX}\} \cup \{\text{all 1-qubit gates}\}$ is exactly universal.

Example. The collection

$$\{\mathsf{H}, \mathsf{T} = \mathsf{P}_{\pi/4}, \mathsf{CX}\}$$

is approximately universal.

Similar to the case of classical computation, we can define the following complexity class:

Definition (**BQP**)**.** The complexity class **BQP** (bounded error, quantum polynomial time) is the class of all decision problems computable with polynomial quantum circuits with at least 2/3 probability of being correct.

We can show that **BQP** is independent of choice of approximately universal gate set. This is not as trivial as the classical case, since when we switch to a different set, we cannot just replace a gate with an equivalent circuit — we can only do so approximately, and we have to make sure we control the error appropriately to maintain the bound of 2/3.

We will consider **BQP** to be the feasible computations with quantum computations.

It is also a fact that **BPP** is a subset of **BQP**. This, again, is not a trivial result. In a quantum computation, we act by unitary matrices, which are invertible. However, boolean functions in classical computing are not invertible in general. So there isn't any straightforward plug-in replacement.

However, it turns out that for any classical computation, there is an equivalent computation that uses reversible/invertible boolean gates, with a modest (i.e. polynomial) overhead of both space and time resources. Indeed, let $f : B_m \to B_n$ be a boolean function. We consider the function

$$\begin{aligned}\tilde{f} : B_{m+n} &\to B_{m+n}\\ (x, y) &\mapsto (x, y \oplus f(x)),\end{aligned}$$

where $\oplus$ is the bitwise addition (i.e. addition in $(\mathbb{Z}/2\mathbb{Z})^n$, e.g. $011 \oplus 110 = 101$). We call x and y the *input register* and *output register* respectively.

Now if we set $y = 0$, then we get $f(x)$ in the second component of $\tilde{f}$. So we can easily obtain f from $\tilde{f}$, and vice versa.

Lemma. For any boolean function $f : B_m \to B_n$, the function

$$\begin{aligned}\tilde{f} : B_{m+n} &\to B_{m+n}\\ (x, y) &\mapsto (x, y \oplus f(x)),\end{aligned}$$

is invertible, and in fact an involution, i.e. is its own inverse.

Proof. Simply note that $x \oplus x = 0$ for any x, and bitwise addition is associative. □

So we can just consider boolean functions that are invertible. There is an easy way of making this a unitary matrix.

Lemma. Let $g : B_k \to B_k$ be a reversible permutation of k-bit strings. Then the linear map on $\mathbb{C}^k$ defined by

$$A : |x\rangle \mapsto |g(x)\rangle$$

on k qubits is unitary.

Proof. This is because the xth column of the matrix of A is in fact $A|x\rangle = |g(x)\rangle$, and since g is bijective, the collection of all $|g(x)\rangle$ are all orthonormal. □

Thus, given any $f : B_m \to B_n$, we get the $n + m$-qubit unitary matrix denoted by U_f, given by

$$U_f |x\rangle |y\rangle = |x\rangle |y \oplus f(x)\rangle .$$

In particular, if we set $|y\rangle = |0\cdots 0\rangle$, then we get

$$|x\rangle\,|0\cdots 0\rangle \xmapsto{U_f} |x\rangle\,|f(x)\rangle\ ,$$

which we can use to evaluate $f(x)$.

What does quantum computation give us? Our gate U_f is unitary, and in particular acts linearly on the states. So by linearity, we have

$$\frac{1}{\sqrt{2^n}}\sum_x |x\rangle\,|0\cdots 0\rangle \xmapsto{U_f} \frac{1}{\sqrt{2^n}}\sum_x |x\rangle\,|f(x)\rangle\ .$$

Now *one* run of U_f gives us this state that embodies all exponentially many values of $f(x)$'s. Of course, we still need to figure out how we can extract useful information from this mixed state, and we will figure that out later on.

While the state

$$|\psi\rangle = \frac{1}{\sqrt{2^n}}\sum_x |x\rangle$$

has exponentially many terms, it can be made in polynomial (and in fact linear) time by n applications of H. Indeed, recall that H is given by

$$|0\rangle \xmapsto{\mathsf{H}} \tfrac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$$

So for an n-qubit state, we have

$$|0\rangle\cdots|0\rangle \xmapsto{\mathsf{H}\otimes\cdots\otimes\mathsf{H}} \frac{1}{\sqrt{2^n}}(|0\rangle + |1\rangle)\cdots(|0\rangle + |1\rangle)\ ,$$

and expanding the right hand side gives us exactly what we want.

3 Some quantum algorithms

3.1 Balanced vs constant problem

We are going to come to our first quantum algorithm.

Here our computational task is a bit special. Instead of an input $i_1, \cdots, i_n \in B_n$, we are given a *black box/oracle* that computes some $f : B_m \to B_n$. We may have some *a priori* promise on f, and we want to determine some property of the function f. The only access to f is querying the oracle with its inputs.

The use of f (classical) or U_f (quantum) counts as one step of computation. The *query complexity* of this task is the least number of times the oracle needs to be queried. Usually, we do not care much about how many times the other gates are used.

Obviously, if we just query all the values of the function, then we can determine anything about the function, since we have complete information. So the goal is to see if we can do it with fewer queries.

The problem we are going to look at is the *balanced vs constant problem*. The input black box is a function $f : B_n \to B_1$. The promise is that f is either

(i) a *constant* function, i.e. $f(x) = 0$ for all x, or $f(0) = 1$ for all x; or

(ii) a *balanced* function, i.e. exactly half of the values in B_n are sent to 1.

We want to determine if f is (i) or (ii) with certainty.

Classically, if we want to find the answer *with certainty*, in the worst case scenario, we will have to perform $2^{n-1} + 1$ queries — if you are really unlucky, you might query a balanced function 2^{n-1} times and get 0 all the time, and you can't distinguish it from a constant 0 function.

Quantumly, we have the *Deutsch-Jozsa algorithm*, that answers the question in 1 query!

A trick we are going to use is something known as "*phase kickback*". Instead of encoding the result as a single bit, we encode them as $\pm$ signs, i.e. as phases of the quantum bits. The "kickback" part is about using the fact that we have

$$|a\rangle \left(e^{i\theta} |b\rangle\right) = \left(e^{i\theta} |a\rangle\right) |b\rangle ,$$

So we might do something to $|b\rangle$ to give it a phase, and then we "kick back" the phase on $|a\rangle$, and hopefully obtain something when we measure $|a\rangle$.

Recall that we have

$$U_f |x\rangle |y\rangle = |x\rangle |y \oplus f(x)\rangle .$$

Here $|x\rangle$ has n qubits, and $|y\rangle$ has 1 qubit.

The non-obvious thing to do is to set the output register to

$$|\alpha\rangle = \frac{|0\rangle - |1\rangle}{2} = \mathsf{H} |1\rangle = \mathsf{HX} |0\rangle .$$

We then note that U_f acts by

$$\begin{aligned} |x\rangle \left(\frac{|0\rangle - |1\rangle}{2}\right) \mapsto |x\rangle \frac{|f(x)\rangle - |1 \oplus f(x)\rangle}{\sqrt{2}} \\ = \begin{cases} |x\rangle \frac{|0\rangle - |1\rangle}{\sqrt{2}} & \text{if } f(x) = 0 \\ |x\rangle \frac{|1\rangle - |0\rangle}{\sqrt{2}} & \text{if } f(x) = 1 \end{cases} \\ = (-1)^{f(x)} |x\rangle |\alpha\rangle . \end{aligned}$$

Now we do this to the superposition over all possible x:

$$\frac{1}{\sqrt{2^n}} \sum |x\rangle |\alpha\rangle \mapsto \left(\frac{1}{\sqrt{2^n}} \sum (-1)^{f(x)} |x\rangle \right) |\alpha\rangle .$$

So one query gives us

$$|\xi_f\rangle = \frac{1}{\sqrt{2^n}} \sum (-1)^{f(x)} |x\rangle .$$

The key observation now is simply that if f is constant, then all the signs are the same. If x is balanced, then exactly half of the signs are $+$ and $-$. The crucial thing is that $|\xi_{f_{\text{const}}}\rangle$ is orthogonal to $|\xi_{f_{\text{balanced}}}\rangle$. This is a good thing, since orthogonality is something we can perfectly distinguish with a quantum measurement.

There is a slight technical problem here. We allow only measurements in the standard $|0\rangle$, $|1\rangle$ basis. So we need want to "rotate" our states to the standard basis. Fortunately, recall that

$$|0\rangle \cdots |0\rangle \xrightarrow{\mathsf{H}\otimes\cdots\otimes\mathsf{H}} \tfrac{1}{\sqrt{2^n}} \sum |x\rangle \ ,$$

Now recall that H is self-inverse, so $H^2 = I$. Thus, if we apply $\mathsf{H} \otimes \cdots \otimes \mathsf{H}$ to $\frac{1}{\sqrt{2^n}} \sum |x\rangle$, then we obtain $|0\rangle \cdots |0\rangle$.

We write

$$|\eta_f\rangle = \mathsf{H} \otimes \cdots \otimes \mathsf{H} \, |\xi_f\rangle .$$

Since H is unitary, we still have

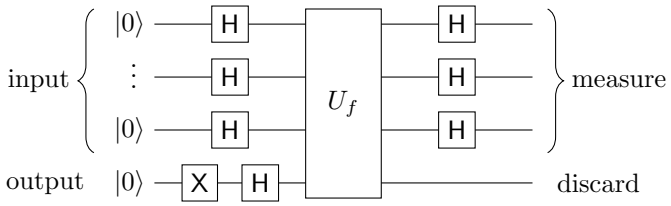
$$|\eta_{f_{\text{const}}}\rangle \perp |\eta_{f_{\text{balanced}}}\rangle .$$

Now we note that if f is constant, then

$$\eta_{f_{\text{const}}} = \pm |0\rangle \cdots |0\rangle .$$

If we look at what $|\eta_{f_{\text{balanced}}}\rangle$ is, it will be a huge mess, but it doesn't really matter — all that matters is that it is perpendicular to $|0\rangle \cdots |0\rangle$.

Now when we measure η_f, if f is a constant function, then we obtain $0 \cdots 0$ with probability 1. If it is balanced, then we obtain something that is not 0 with probability 1. So we can determine the result with probability 1.



This uses exactly one query, with $1 + (n + 1) + n + n = O(n)$ elementary gates and measurements.

What if we tolerate error in the balanced vs constant problem? In other words, we only require that the answer is correct with probability $1 - \varepsilon$ with $0 < \varepsilon < \frac{1}{2}$.

In the quantum case, nothing much changes, since we are probably not going to do better than 1 query. However, we no longer have a huge benefit over classical algorithms. There is a classical randomized algorithm with $O(\log(1/\varepsilon))$ queries, and in particular does not depend on n.

Indeed, we do it the obvious way — we choose some K x-values uniformly at random from B_n, say $x_1, \cdots, x_n$ (where K is fixed and determined later). We then evaluate $f(x_1), \cdots, f(x_n)$.

If all the outputs are the same, then we say f is constant. If they are not the same, then we say f is balanced.

If f actually is constant, then the answer is correct with probability 1. If f is balanced, then each $f(x_i)$ is 0 or 1 with equal probability. So the probability of getting the same values for all x_i is

$$\frac{2}{2^K} = 2^{1-K}.$$

This is our failure probability. So if we pick

$$K > \log_2(\varepsilon^{-1}) + 1,$$

then we have a failure probability of less than ε.

Can we decide *every* yes/no question about $f : B_n \to B_1$'s by quantum algorithms with "a few" queries? The answer is no. One prominent example is the *SAT problem* (*satisfiability problem*) — given an arbitrary f, we want to determine if there an x such that $f(x) = 1$? It can be shown that any quantum algorithm (even if we allow for bounded errors) needs at least $O(\sqrt{2^n})$ queries, which is achieved by Grover's algorithm. Classically, we need $O(2^n)$ queries. So we have achieved a square root speedup, which is good, but not as good as the Deutsch-Jozsa algorithm.

In any case, the Deutsch-Jozsa algorithm demonstrates how we can achieve an exponential benefit with quantum algorithms, but it happens only when we have no error tolerance. In real life scenario, external factors will lead to potential errors anyway, and requiring that we are always correct is not a sensible requirement.

There are other problems where quantum algorithms are better:

Example (Simon's algorithm)**.** The *Simon's problem* is a promise problem about $f : B_n \to B_n$ with provably exponential separation between classical ($O(2^{n/4})$) and quantum ($O(n)$) query complexity even with bounded error. The details are on the first example sheet.

3.2 Quantum Fourier transform and periodicities

We've just seen some nice examples of benefits of quantum algorithms. However, oracles are rather unnatural problems — it is rare to just have a black-box access to a function without knowing anything else about the function.

How about more "normal" problems? The issue with trying to compare quantum and classical algorithms for "normal" problems is that we don't actually have any method to find the lower bound for the computation complexity. For example, while we have not managed to find polynomial prime factorization algorithms, we cannot prove for sure that there isn't any classical algorithm that is polynomial time. However, for the prime factorization problem, we *do* have a

quantum algorithm that does much better than all known classical algorithms. This is *Shor's algorithm*, which relies on the toolkit of the quantum Fourier transform.

We start by defining the quantum Fourier transform.

Definition (Quantum Fourier transform mod N)**.** Suppose we have an N-dimensional state space with basis $|0\rangle, |1\rangle, \cdots, |N-1\rangle$ labelled by $\mathbb{Z}/N\mathbb{Z}$. The *quantum Fourier transform mod N* is defined by

$$\mathsf{QFT} : |a\rangle \mapsto \frac{1}{\sqrt{N}} \sum_{b=0}^{N-1} e^{2\pi i ab/N} |b\rangle .$$

The matrix entries are

$$[\mathsf{QFT}]_{ab} = \frac{1}{\sqrt{N}} \omega^{ab}, \quad \omega = e^{2\pi i/N},$$

where $a, b = 0, 1, \cdots, N-1$. We write QFT_n for the quantum Fourier transform mod n.

Note that we are civilized and start counting at 0, not 1.
We observe that the matrix $\sqrt{N}\mathsf{QFT}$ is

(i) Symmetric

(ii) The first (i.e. 0th) row and column are all 1's.

(iii) Each row and column is a geometric progression $1, r, r^2, \cdots, r^{n-1}$, where $r = \omega^k$ for the kth row or column.

Example. If we look at QFT_2, then we get our good old H. However, QFT_4 is not $H \otimes H$.

Proposition. QFT is unitary.

Proof. We use the fact that

$$1 + r + \cdots + r^{N-1} = \begin{cases} \frac{1-r^N}{1-r} & r \neq 1 \\ N & r = 1 \end{cases}.$$

So if $r = \omega^k$, then we get

$$1 + r + \cdots + r^{N-1} = \begin{cases} 0 & k \not\equiv 0 \mod N \\ N & k \equiv 0 \mod N \end{cases}.$$

Then we have

$$(\mathsf{QFT}^\dagger \mathsf{QFT})_{ij} = \frac{1}{\sqrt{N}^2} \sum_k \omega^{-ik} \omega^{jk} = \frac{1}{N} \sum_k \omega^{(j-i)k} = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}. \qquad \square$$

We now use the quantum Fourier Transform to solve the periodicity problem.

Example. Suppose we are given $f : \mathbb{Z}/N\mathbb{Z} \to Y$ (for some set Y). We are promised that f is periodic with some period $r \mid N$, so that

$$f(x + r) = f(x)$$

for all x. We also assume that f is injective in each period, so that

$$0 \leq x_1 \neq x_2 \leq r - 1 \quad \text{implies} \quad f(x_1) \neq f(x_2).$$

The problem is to find r, with any constant level of error $1 - \varepsilon$ independent of N. Since this is not a decision problem, we can allow $\varepsilon > \frac{1}{2}$.

In the classical setting, if f is viewed as an oracle, then $O(\sqrt{N})$ queries are necessary and sufficient. We are going to show that quantumly, $O(\log \log N)$ queries with $O(\text{poly}(\log N))$ processing steps suffice. In later applications, we will see that the relevant input size is $\log N$, not N. So the classical algorithm is exponential time, while the quantum algorithm is polynomial time.

Why would we care about such a problem? It turns out that later we will see that we can reduce prime factorization into a periodicity problem. While we will actually have a very explicit formula for f, there isn't much we can do with it, and treating it as a black box and using a slight modification of what we have here will be much more efficient than any known classical algorithm.

The quantum algorithm is given as follows:

(i) Make $\frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle$. For example, if $N = 2^n$, then we can make this using $\mathsf{H} \otimes \cdots \otimes \mathsf{H}$. If N is not a power of 2, it is not immediately obvious how we can make this state, but we will discuss this problem later.

(ii) We make one query to get

$$|f\rangle = \frac{1}{\sqrt{N}} \sum |x\rangle |f(x)\rangle .$$

(iii) We now recall that $r \mid N$, Write $N = Ar$, so that A is the number of periods. We measure the second register, and we will see some $y = f(x)$. We let x_0 be the *least* x with $f(x) = y$, i.e. it is in the first period. Note that we don't know what x_0 is. We just know what y is.

By periodicity, we know there are exactly A values of x such that $f(x) = y$, namely

$$x_0,\ x_0 + r,\ x_0 + 2r,\ \cdots,\ x_0 + (A - 1)r.$$

By the Born rule, the first register is collapsed to

$$|\text{per}\rangle = \left(\frac{1}{\sqrt{A}} \sum_{j=0}^{A-1} |x_0 + jr\rangle \right) |f(x_0)\rangle .$$

We throw the second register away. Note that x_0 is chosen randomly from the first period $0, 1, \cdots, r - 1$ with equal probability.

What do we do next? If we measure $|\text{per}\rangle$, we obtain a random j-value, so what we actually get is a random element (x_0th) of a random period (jth), namely a uniformly chosen random number in $0, 1, \cdots, N$. This is not too useful.

(iv) The solution is the use the quantum Fourier transform, which is not surprising, since Fourier transforms are classically used to extract periodicity information.

Apply QFT_N to $|\text{per}\rangle$ now gives

$$\begin{aligned}\mathsf{QFT}_N |\text{per}\rangle &= \frac{1}{\sqrt{NA}} \sum_{j=0}^{n-1} \sum_{y=0}^{N-1} \omega^{(x_0+jr)y} |y\rangle \\ &= \frac{1}{\sqrt{NA}} \sum_{y=0}^{N-1} \omega^{x_0 y} \left(\sum_{j=0}^{n-1} \omega^{jry} \right) |y\rangle\end{aligned}$$

We now see the inner sum is a geometric series. If $\omega^{ry} = 1$, then this sum is just A. Otherwise, we have

$$\sum_{j=0}^{A-1} \omega^{jry} = \frac{1-\omega^{rA}}{1-\omega^{ry}} = \frac{1-1}{1-\omega^{ry}} = 0.$$

So we are left with

$$\mathsf{QFT}_n |\text{per}\rangle = \sqrt{\frac{A}{N}} \sum_{k=0}^{r-1} \omega^{x_0 kN/r} \left| k\frac{N}{r} \right\rangle.$$

Note that before the Fourier transform, the random shift of x_0 lied in the label $|x_0 + jr\rangle$. After the Fourier transform, it is now encoded in the phase instead.

(v) Now we can measure the label, and we will get some C which is a multiple $k_0 \frac{N}{r}$, where $0 \leq k_0 \leq r-1$ is chosen uniformly at random. We rewrite this equation as

$$\frac{k_0}{r} = \frac{C}{N}.$$

We know C, because we just measured it, and N is a given in the question. Also, k_0 is randomly chosen, and r is what we want. So how do we extract that out?

If by good chance, we have k_0 coprime to r, then we can cancel C/N to lowest terms and read off r as the resulting denominator $\tilde{r}$. Note that cancelling C/N to lowest terms can be done quickly by Euclidean algorithm. But how likely are we to be so lucky? We can just find some number theory book, and figure out that the number of natural numbers $< r$ that are coprime to r grows as $O(r/\log\log r)$. More precisely, it is $\sim e^{-\gamma} r/\log\log r$, where γ is the other Euler's constant. We note that

$$O\left(\frac{r}{\log\log r}\right) > O\left(\frac{1}{\log\log N}\right).$$

So if k_0 is chosen uniformly and randomly, the probability that k_0 is coprime to r is at least $O(1/\log\log N)$.

Note that if k_0 is *not* coprime with r, then we have $\tilde{r} \mid r$, and in particular $\tilde{r} < r$. So we can check if $\tilde{r}$ is a true period — we compute $f(0)$ and $f(\tilde{r})$,

and see if they are the same. If $\tilde{r}$ is wrong, then they cannot be equal as f is injective in the period.

While the probability of getting a right answer decreases as $N \to \infty$, we just have to do the experiment many times. From elementary probability, if an event has some (small) success probability p, then given any $0 < 1 - \varepsilon < 1$, for $M = -\frac{\log \varepsilon}{p}$ trials, the probability that there is at least one success is $> 1 - \varepsilon$. So if we repeat the quantum algorithm $O(\log \log N)$ times, and check $\tilde{r}$ each time, then we can get a true r with any constant level of probability.

(vi) We can further improve this process — if we have obtained two attempts $\tilde{r}, \tilde{r}'$, then we know r is at least their least common multiple. So we can in fact achieve this in constant time, if we do a bit more number theory. However, the other parts of the algorithm (e.g. cancelling C/N down to lowest terms) still use time polynomial in $\log N$. So we have a polynomial time algorithm.

There is one thing we swept under the carpet. We need to find an efficient way of computing the quantum Fourier transform, or else we just hid all our complexity in the quantum Fourier transform.

In general, we would expect that a general unitary operations on n qubits needs $\exp(n)$ elementary circuits. However, the quantum Fourier transform is special.

Fact. QFT_{2^n} can be implemented by a quantum circuit of size $O(n^2)$.

The idea of the construction is to mimic the classical fast Fourier transform. An important ingredient of it is:

Fact. The state

$$\mathsf{QFT}_{2^n} |x\rangle = \frac{1}{2^{n/2}} \sum_{y=0}^{2^n-1} \omega^{xy} |y\rangle$$

is in fact a product state.

We will not go into the details of implementation.

We can generalize the periodicity problem to arbitrary groups, known as the *hidden subgroup problem*. We are given some oracle for $f : G \to Y$, and we are promised that there is a subgroup $H < G$ such that f is constant and distinct on cosets of H in G. We want to find H (we can make "find" more precise in two ways — we can either ask for a set of generators, or provide a way of sampling uniformly from H).

In our case, we had $G = (\mathbb{Z}/N\mathbb{Z}, +)$, and our subgroup was

$$H = \{0, r, 2r, \cdots, (A-1)r\}.$$

Unfortunately, we do not know how to do this efficiently for a group in general.

3.3 Shor's algorithm

All that was a warm up for *Shor's algorithm*. This is a quantum algorithm that factorizes numbers in polynomial time. The crux of the algorithm will be a modified version of the quantum Fourier transform.

The precise statement of the problem is as follows — given an integer N with $n = \log N$ digits, we want to find a factor $1 < K < N$. Shor's algorithm will achieve this with constant probability $(1 - \varepsilon)$ in $O(n^3)$ time. The best known classical algorithm is $e^{O(n^{1/3}(\log n)^{2/3})}$.

To do this, we will use the periodicity algorithm. However, there is one subtlety involved. Instead of working in $\mathbb{Z}/n\mathbb{Z}$, we need to work in $\mathbb{Z}$. Since computers cannot work with infinitely many numbers, we will have to truncate it somehow. Since we have no idea what the period of our function will be, we must truncate it randomly, and we need to make sure we can control the error introduced by the truncation.

We shall now begin. Given an N, we first choose some $1 < a < N$ uniformly randomly, and compute $\text{hcf}(a, N)$. If it is not equal to 1, then we are finished. Otherwise, by Euler's theorem, there is a least power r of a such that $a^r \equiv 1 \bmod N$. The number r is called the *order* of a mod N. It follows that the function $f : \mathbb{Z} \to \mathbb{Z}/N\mathbb{Z}$ given by $f(k) = a^k \bmod N$ has period r, and is injective in each period.

Note that $f(k)$ can be efficiently computed in $\text{poly}(\log k)$ time, by repeated squaring. Also note that classically, it is hard to find r, even though f has a simple formula!

It was known to Legendre in 1800 that knowing r means we can factor n. Suppose we can find r, and further suppose r is even. Then we have

$$a^r - 1 \equiv (a^{r/2} + 1)(a^{r/2} - 1) \equiv 0 \pmod{N}.$$

So N exactly divides the product. By minimality of r, we know N does not divide $a^{r/2} - 1$. So if N does not divide $a^{r/2} + 1$ as well, then $\text{hcf}(N, a^{r/2} \pm 1)$ are non-trivial factors of N.

For this to work, we needed two assumptions – r is even, and $a^{r/2} \not\equiv -1 \pmod{N}$. Fortunately, there is a theorem in number theory that says if N is odd and not a prime power, and a is chosen uniformly at random, then the probability that these two things happen is at least $\frac{1}{2}$. In fact, it is $\geq 1 - \frac{1}{2^{m-1}}$, where m is the number of prime factors of N.

So if we repeat this k times, the probability that they all fail to give a factor is less than $\frac{1}{2^k}$. So this can be as small as we wish.

What about the other possibilities? If N is even, then we would have noticed by looking at the last digit, and we can just write down 2. If $N = c^\ell$ for $c, \ell > 2$, then there is a classical polynomial time algorithm that outputs c, which is a factor. So these are the easy cases.

Everything we've done so far is classical! The quantum part comes in when we want to compute r. We know that $f(k) = a^k$ is periodic on $\mathbb{Z}$, which is an infinite domain. So we cannot just apply our periodicity algorithm.

By number theory, we know that r is at most N. But other than that, we have no idea what r actually is, nor do we know of any multiple of r. So we cannot apply the periodicity argument directly. Instead, we pick a big number 2^m, and work on the domain $D = \{0, 1, \cdots, 2^m - 1\} = \mathbb{Z}/2^m\mathbb{Z}$. How do we choose m? The idea is that we want $0, \cdots, 2^m - 1$ to contain B full periods, plus some extra "corrupt" noise b, so

$$2^m = Br + b,$$

with $0 \leq b < r$. Since we want to separate out the periodicity information from the corrupt noise, we will want b to be relatively small, compared to Br. We

know the size of b is bounded by r, hence by N. So we need 2^m to be "much larger" than N. It turns out picking $2^m > N^2$ is enough, and we will pick m to be the smallest number such that this holds.

We now study the effect of corruption on the periodicity algorithm. We again make the state

$$|f\rangle = \frac{1}{\sqrt{2^m}} \sum |x\rangle\,|f(x)\rangle\,.$$

and measure the value of f. We then get

$$|\text{per}\rangle = \frac{1}{\sqrt{A}} \sum_{k=0}^{A-1} |x_0 + kr\rangle\,,$$

where $A = B$ or $B + 1$, depending on whether $x_0 \leq b$ or not. As before, we apply QFT_{2^m} to obtain

$$\mathsf{QFT}_{2^m}\,|\text{per}\rangle = \sum_{c=0}^{2^n-1} \tilde{f}(c)\,|c\rangle\,.$$

When we did this before, with an exact period, most of the $\hat{f}(c)$ is zero. However, this time things are a bit more messy. As before, we have

$$\tilde{f}(c) = \frac{\omega^{cx_0}}{\sqrt{A}\sqrt{2^m}}[1 + \alpha + \cdots + \alpha^{A-1}], \quad \alpha = e^{2\pi i c r/2^m}.$$

The important question is, when we measure this, which c's will we see with "good probability"? With exact periodicity, we knew that $\frac{2^m}{r} = A$ is an exact integer. So $\tilde{f}(c) = 0$ except when c is a multiple of A. Intuitively, we can think of this as interference, and we had totally destructive and totally constructive interference respectively.

In the inexact case, we will get constructive interference for those c such that the phase α is close to 1. These are the c's with $\frac{cr}{2^m}$ nearest to integers k, and the powers up to α^{A-1} don't spread too far around the unit circle. So we avoid cancellations.

So we look at those special c's having this particular property. As c increases from 0 to $2^m - 1$, the angle $\frac{cr}{2^m}$ increments by $\frac{r}{2^m}$ each time from 0 up to r. So we have c_k's for each $k = 0, 1, \cdots, r - 1$ such that

$$\left|\frac{c_k r}{2^m} - k\right| < \frac{1}{2} \cdot \frac{r}{2^m}.$$

In other words, we have

$$\left|c_k - k\frac{2^m}{r}\right| < \frac{1}{2}.$$

So the c_k are the integers nearest to the multiples of $2^m/r$.

In $\tilde{f}(c)$, the α's corresponding to the c_k's have the smallest phases, i.e. nearest to the positive real axis. We write

$$\frac{c_k r}{2^m} = k + \xi,$$

where

$$k \in \mathbb{Z}, \quad |\xi| < \frac{1}{2}\frac{r}{2^m}.$$

Then we have

$$\alpha^n = \exp\left(2\pi i \frac{c_k r}{2^m} n\right) = \exp\left(e\pi i(k+\xi)n\right) = \exp(2\pi i \xi n)$$

Now for $n < A$, we know that $|2\xi n| < \pi$, and thus $1, \alpha, \alpha^2, \cdots, \alpha^{A-1}$ all lie in the lower half plane or upper half plane.

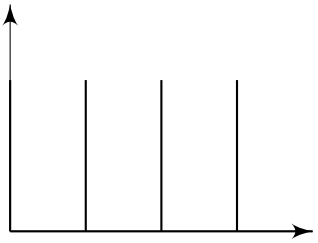
Doing all the algebra needed, we find that if $\mathsf{QFT}\,|\text{per}\rangle$ is measured, then for any c_k as above, we have

$$\text{Prob}(c_k) > \frac{\gamma}{r},$$

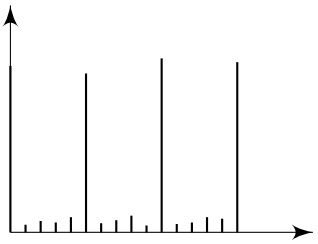
where

$$\gamma = \frac{4}{\pi^2} \approx 0.4.$$

Recall that in the exact periodicity case, the points c_k hit the integers exactly, and instead of γ we had 1. The distribution of the c's then look like:



With inexact periods, we obtain something like



Now how do we get r from a c_k? We know

$$\left|\frac{c_k}{2^m} - \frac{k}{r}\right| < \frac{1}{2^{m+1}} < \frac{1}{2N^2}.$$

We claim that there is at most 1 fraction $\frac{k}{r}$ with denominator $< N$ such that this inequality holds. So this inequality does uniquely determine k/r.

Indeed, suppose $\frac{k}{r}$ and $\frac{k'}{r'}$ both work. Then we have

$$\left|\frac{k'}{r'} - \frac{k}{r}\right| = \frac{|k'r - r'k|}{rr'} > \frac{1}{rr'} > \frac{1}{N^2}.$$

However, we also have

$$\left|\frac{k'}{r'} - \frac{k}{r}\right| \leq \left|\frac{k'}{r'} - \frac{c_k}{2^m}\right| + \left|\frac{c_k}{2^m} - \frac{k}{r}\right| < \frac{1}{2N^2} + \frac{1}{2N^2} = \frac{1}{N^2}.$$

So it follows that we must have $\frac{k'}{r'} = \frac{k}{r}$.

We introduce the notion of a "good" c_k value, which is when k is coprime to r. The probability of getting a good c_k is again

$$O(1/\log\log r) > O(1/\log\log N).$$

Note that this is the same rate as the case of exact periodicity, since we have only lost a constant factor of γ! If we did have such a c_k, then now r is uniquely determined.

However, there is still the problem of finding r from a good c_k value. At this point, this is just classical number theory.

We can certainly try all $\frac{k'}{r'}$ with $k' < r' < N$ and find the closest one to $c_k/2^m$, but there are $O(N^2)$ fractions to try, but we want a $O(\text{poly}(\log N))$ algorithm. Indeed, if we were to do it this way, we might as well try all numbers less than N and see if they divide N. And this is just $O(N)$!

The answer comes from the nice theory of continued fractions. Any rational number $\frac{s}{t} < 1$ has a continued fraction expansion

$$\frac{s}{t} = \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \cdots}}}.$$

Indeed to do this, we simply write

$$\frac{s}{t} = \frac{1}{\frac{t}{s}} = \cfrac{1}{a_1 + \frac{s_1}{t_1}},$$

where we divide t by s to get $t = a_1 s + s_1$, and then put $t_1 = s$. We then keep going on with $\frac{s_1}{t_1}$. Since the numbers s_i, t_i keep getting smaller, it follows that this process will eventually terminate.

Since it is very annoying to type these continued fractions in LaTeX, we often write the continued fraction as

$$\frac{s}{t} = [a_1, a_2, a_3, \cdots, a_n].$$

We define the kth convergent of $\frac{s}{t}$ to be

$$\frac{p_k}{q_k} = [a_1, a_2, \cdots, a_k].$$

There are some magic results from number theory that gives us a simple recurrence relation for the convergents.

Lemma. For $a_1, a_2, \cdots, a_\ell$ any positive reals, we set

$$p_0 = 0 \qquad q_0 = 1$$
$$p_1 = 1 \qquad q_1 = a_1$$

We then define

$$p_k = a_k p_{k-1} + p_{k-2}$$
$$q_k = a_k q_{k-1} + q_{k-2}$$

Then we have

(i) We have
$$[a_1, \cdots, a_k] = \frac{p_k}{q_k}.$$

(ii) We also have
$$q_k p_{k-1} - p_k q_{k-1} = (-1)^k.$$
In particular, p_k and q_k are coprime.

From a bit more number theory, we find that

Fact. If $s < t$ are m-bit integers, then the continued fraction has length $O(m)$, and all convergents $\frac{p_k}{q_k}$ can be computed in $O(m^3)$ time.

More importantly, we have the following result:

Fact. Let $0 < x < 1$ be rational, and suppose $\frac{p}{q}$ is rational with
$$\left|x - \frac{p}{q}\right| < \frac{1}{2q^2}.$$
Then $\frac{p}{q}$ is a convergent of the continued fraction of x.

Then by this theorem, for a good c, we know $\frac{k}{r}$ must be a convergent of $\frac{c}{2^m}$. So we compute all convergents find a (unique) one whose denominator is less than N and is within $\frac{1}{2N^2}$ of $\frac{c}{2^m}$. This gives us the value of r, and we are done.

In fact, this last classical part is the slowest part of the algorithm.

Example. Suppose we want to factor $N = 39$. Suppose the random a we chose is $a = 7 < 39$, which is coporime to N. Let r be the period of $f(x) = 7^x \bmod 39$.

We notice
$$1024 = 2^{10} < N^2 = 1621 < 2^{11} = 2048.$$
So we pick $m = 11$. Suppose the measurement of $\mathsf{QFT}_{2^m} |\text{per}\rangle$ yeilds $c = 853$.

By the theory, this has a constant probability (approximately 0.4) to satisfy
$$\left|\frac{853}{2^m} - \frac{k}{r}\right| < \frac{1}{2^{m+1}} = \frac{1}{2^{12}} < \frac{1}{2N^2}.$$
We also have a probability of $O(1/\log\log r)$ to have k and r coprime. In this case, c is indeed "good". So there is a unique $\frac{k}{r}$ satisfying
$$\left|\frac{853}{2048} - \frac{k}{r}\right| < \frac{1}{2^{12}}.$$
So to find $\frac{k}{r}$, we do the continued fraction expansion of $\frac{853}{2048}$. We have
$$\frac{853}{2048} = \frac{1}{\frac{2048}{853}} = \frac{1}{2 + \frac{342}{853}} = \frac{1}{2 + \frac{1}{\frac{853}{342}}} = \frac{1}{2 + \frac{1}{2 + \frac{169}{342}}} = \cdots = [2, 2, 2, 42, 4].$$

We can then compute the convergents

$$\begin{aligned}[2] &= \frac{1}{2}\\ [2,2] &= \frac{2}{5}\\ [2,2,2] &= \frac{5}{12}\\ [2,2,2,42] &= \frac{212}{509}\\ [2,2,2,42,4] &= \frac{853}{2048}\end{aligned}$$

Of all these numbers, only $\frac{5}{12}$ is within $\frac{1}{2^{12}}$ of $\frac{853}{2048}$ and whose denominator is less than $N = 39$.

If we do not assume k and r are coprime, then the possible $\frac{k}{r}$ are

$$\frac{5}{12}, \frac{10}{24}, \frac{15}{36}.$$

If we assume that $\frac{k}{r}$ are coprime, then $r = 12$. Indeed, we can try that

$$7^{12} \equiv 1 \pmod{39}.$$

So we now know that

$$39 \mid (7^6 + 1)(7^6 - 1).$$

We now hope/expect with probability $> \frac{1}{2}$ exactly that it goes partly into each factor. We can compute

$$\begin{aligned}7^6 + 1 &= 117650 \equiv 26 \pmod{39}\\ 7^6 - 1 &= 117648 \equiv 24 \pmod{39}\end{aligned}$$

We can then compute

$$\mathrm{hcf}(26, 39) = 13, \quad \mathrm{hcf}(24, 39) = 3 \pmod{39}.$$

We see that 3 and 13 are factors of 39.

3.4 Search problems and Grover's algorithm

We are now going to turn our attention to search problems. These are very important problems in computing, as we can formulate almost all problems as some sort of search problems.

One important example is simultaneous constraint satisfaction. Here we have a large configuration space of options, and we want to find some configuration that satisfies some constraints. For example, when designing a lecture timetable for Part III courses, we need to schedule the courses so that we don't clash two popular courses in the same area, and the courses need to have big enough lecture halls, and we have to make sure a lecturer doesn't have to simultaneously lecture two courses at the same time. This is very complicated.

In general, search problems have some common features:

(i) Given any instance of solution attempt, it is easy to check if it is good or not.

(ii) There are exponentially many possible instances to try out.

One example is the boolean satisfiability problem, which we have already seen before.

Example (Boolean satisfiability problem)**.** The *boolean satisfiability problem* (SAT) is as follows — given a Boolean formula $f : B_n \to B$, we want to know if there is a "satisfying argument", i.e. if there is an x with $f(x) = 1$.

This has complexity class **NP**, standing for *non-deterministic polynomial time*. There are many ways to define **NP**, and here we will provide two. The first definition of **NP** will involve the notion of a verifier:

Definition (Verifier)**.** Suppose we have a language $L \subseteq B^*$, where

$$B^* = \bigcup_{n \in \mathbb{N}} B_n$$

is the set of all bit strings.

A *verifier* for L is a computation $V(w, c)$ with two inputs w, c such that

(i) V halts on all inputs.

(ii) If $w \in L$, then for *some* c, $V(w, c)$ halts with "accept".

(iii) If $w \notin L$, then for *all* c, $V(w, c)$ halts with "reject".

A *polynomial time verifier* is a V that runs in polynomial time in $|w|$ (not $|w| + |c|$!).

We can think of c as "certificate of membership". So if you are a member, you can exhibit a certificate of membership that you are in there, and we can check if the certification is valid. However, if you are not a member, you cannot "fake" a certificate.

Definition (Non-deterministic polynomial time problem)**. NP** is the class of languages that have polynomial time verifiers.

Example. The SAT problem is in **NP**. Here c is the satisfying argument, and $V(f, c)$ just computes $f(c)$ and checks whether it is 1.

Example. Determining if a number is composite is in **NP**, where a certificate is a factor of the number.

However, it is not immediately obvious that testing if a number is prime is in **NP**. It is an old result that it indeed is, and recent progress shows that it is in fact in **P**.

It is rather clear that $\mathbf{P} \subseteq \mathbf{NP}$. Indeed, if we can check membership in polynomial time, then we can also construct a verifier in polynomial time that just throws the certificate away and check directly.

There is another model of **NP**, via *non-deterministic computation*. Recall that in probabilistic computation, in some steps, we had to pick a random

number, and picking a different number would lead to a different "branch". In the case of non-deterministic computation, we are allowed to take *all* paths at the same time. If *some* of the paths end up being accepting, then we accept the input. If *all* paths reject, then we reject the input. Then we can alternatively say a problem is in **NP** if there is a polynomial-time non-deterministic machine that checks if the string is in the language.

It is not difficult to see that these definitions of **NP** are equivalent. Suppose we have a non-deterministic machine that checks if a string is in the language. Then we can construct a verifier whose certificate is a prescription of which particular branch we should follow. Then the verifier just takes the prescription, follows the path described and see if we end up being accepted.

Conversely, if we have a verifier, we can construct a non-deterministic machine by testing a string on *all* possible certificates, and check if any of them accepts.

Unfortunately, we don't know anything about how these different complexity classes compare. We clearly have $\mathbf{P} \subseteq \mathbf{BPP} \subseteq \mathbf{BQP}$ and $\mathbf{P} \subseteq \mathbf{NP}$. However, we do not know if these inclusions are strict, or how **NP** compares to the others.

Unstructured search problem and Grover's algorithm

Usually, when we want to search something, the search space we have is structured in some way, and this greatly helps our searching problem.

For example, if we have a phone book, then the names are ordered alphabetically. If we want to find someone's phone number, we don't have to look through the whole book. We just open to the middle of the book, and see if the person's name is before or after the names on the page. By one lookup like this, we have already eliminated half of the phone book we have to search through, and we can usually very quickly locate the name.

However, if we know someone's phone number and want to figure out their name, it is pretty much hopeless! This is the problem with unstructured data!

So the problem is as follows: we are given an unstructured database with $N = 2^n$ items and a *unique* good item (or no good items). We can query any item for good or bad-ness. The problem is to find the good item, or determine if one exists.

Classically, $O(N)$ queries are necessary and sufficient. Even if we are asking for a right result with fixed probability c, if we pick items randomly to check, then the probability of seeing the "good" one in k queries is given by k/N. So we still need $O(N)$ queries for any fixed probability.

Quantumly, we have *Grover's algorithm*. This needs $O(\sqrt{N})$ queries, and this is both necessary and sufficient.

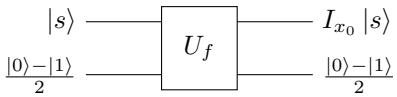
The database of $N = 2^n$ items will be considered as an oracle $f : B_n \to B_1$. it is promised that there is a unique $x_0 \in B_n$ with $f(x_0) = 1$. The problem is to find x_0. Again, we have the quantum version

$$U_f \left|x\right\rangle \left|y\right\rangle = \left|x\right\rangle \left|y \otimes f(x)\right\rangle .$$

However, we'll use instead I_{x_0} on n qubits given by

$$I_{x_0} \left|x\right\rangle = \begin{cases} \left|x\right\rangle & x \neq x_0 \\ -\left|x\right\rangle & x \neq x_0 \end{cases} .$$

This can be constructed from U_f as we've done before, and one use of I_{x_0} can be done with one use of U_f.



We can write I_{x_0} as

$$I_{x_0} = I - 2|x_0\rangle\langle x_0|,$$

where I is the identity operator.

We are now going to state the *Grover's algorithm*, and then later prove that it works.

For convenience, we write

$$\mathsf{H}_n = \underbrace{\mathsf{H} \otimes \cdots \otimes \mathsf{H}}_{n \text{ times}}$$

We start with a uniform superposition

$$|\psi_0\rangle = \mathsf{H}_n|0\cdots 0\rangle = \frac{1}{\sqrt{n}} \sum_{\text{all } x} |x\rangle.$$

We consider the *Grover iteration operator* on n qubits given by

$$\mathcal{Q} = -\mathsf{H}_n I_0 \mathsf{H}_n I_{x_0}.$$

Here running I_{x_0} requires one query (whereas I_0 is "free" because it is just $I - 2|0\rangle\langle 0|$).

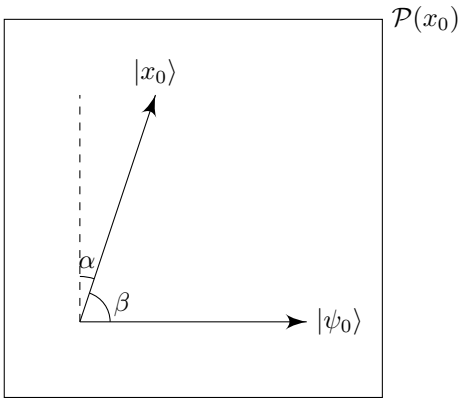
Note that all these operators are all real. So we can pretend we are living in the real world and have nice geometric pictures of what is going on. We let $\mathcal{P}(x_0)$ be the (real) plane spanned by $|x_0\rangle$ and $|\psi_0\rangle$. We claim that

(i) In this plane $\mathcal{P}(x_0)$, this operator $\mathcal{Q}$ is a rotation by 2α, where

$$\sin\alpha = \frac{1}{\sqrt{N}} = \langle x_0|\psi_0\rangle.$$

(ii) In the orthogonal complement $\mathcal{P}(x_0)^\perp$, we have $\mathcal{Q} = -I$.

We will prove these later on. But if we know this, then we can repeatedly apply $\mathcal{Q}$ to $|\psi_0\rangle$ to rotate it near to $|x_0\rangle$, and then measure. Then we will obtain $|x_0\rangle$ with very high probability:



The initial angle is

$$\cos\beta = \langle x_0|\psi_0\rangle = \frac{1}{\sqrt{N}}.$$

So the number of iterations needed is

$$\frac{\cos^{-1}(1/\sqrt{N})}{2\sin^{-1}(1/\sqrt{N})} = \frac{\beta}{2\alpha}.$$

In general, this is not an integer, but applying a good integer approximation to it will bring us to very close to $|x_0\rangle$, and thus we measure x_0 with high probability. For large n, the number of iterations is approximately

$$\frac{\pi/2}{2/\sqrt{N}} = \frac{\pi}{4}\sqrt{N}.$$

Example. Let's do a boring example with $N = 4$. The initial angle satisfies

$$\cos\beta = \frac{1}{\sqrt{4}} = \frac{1}{2}.$$

So we know

$$\beta = \frac{\pi}{3}.$$

Similarly, we have

$$2\alpha = 2\sin^{-1}\frac{1}{2} = \frac{\pi}{3}.$$

So 1 iteration of $\mathcal{Q}$ will rotate $|\psi_0\rangle$ *exactly* to $|x_0\rangle$, so we can find it with *certainty* with 1 lookup.

Now we prove that this thing actually works. In general, for any unitary U and $I_{|\psi\rangle} = I - 2|\psi\rangle\langle\psi|$, we have

$$UI_{|\psi\rangle}U^\dagger = UIU^\dagger - 2U|\psi\rangle\langle\psi|U^\dagger = I_{U|\psi\rangle}.$$

In particular, since H_n is self-adjoint, i.e. $H_n^\dagger H_n$, and that by definition $H_n|0\rangle = |\psi_0\rangle$, we know

$$\mathcal{Q} = -H_n I_0 H_n I_{x_0} = -I_{|\psi_0\rangle} I_{x_0}.$$

Next we note that for any $|\psi\rangle$ and $|\xi\rangle$, we know by definition

$$I_{|\psi\rangle}|\xi\rangle = |\xi\rangle - 2|\psi\rangle\langle\psi|\xi\rangle.$$

So this modifies $|\xi\rangle$ by some multiple of $|\psi\rangle$. So we know our operator

$$\mathcal{Q}|\psi\rangle = -I_{|\psi_0\rangle} I_{x_0}|\psi\rangle$$

modifies $|\psi\rangle$ first by some multiple of $|x_0\rangle$, then by some multiple of ψ_0. So if $|\xi\rangle \in \mathcal{P}(x_0)$, then $\mathcal{Q}|\psi\rangle \in \mathcal{P}(x_0)$ too! So $\mathcal{Q}$ preserves $\mathcal{P}(x_0)$.

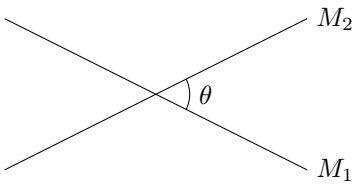
We know that $\mathcal{Q}$ is a unitary, and it is "real". So it must be a rotation or a reflection, since these are the only things in $\mathrm{O}(2)$. We can explicitly figure out what it is. In the plane $\mathcal{P}(x_0)$, we know $I_{|x_0\rangle}$ is reflection in the mirror line perpendicular to $|x_0\rangle$. Similarly, $I_{|\psi_0\rangle}$ is reflection in the mirror line perpendicular to $|\psi_0\rangle$.

We now use the following facts about 2D Euclidean geometry:

(i) If R is a reflection in mirror M along $|M\rangle$, then $-R$ is reflection in mirror $M^\perp$ along $\left|M^\perp\right\rangle$.

To see this, we know any vector can be written as $a\,|M\rangle + b\left|M^\perp\right\rangle$. Then R sends this to $a\,|M\rangle - b\left|M^\perp\right\rangle$, while $-R$ sends it to $-a\,|M\rangle + b\left|M^\perp\right\rangle$, and this is reflection in $\left|M^\perp\right\rangle$.

(ii) Suppose we have mirrors M_1 and M_2 making an angle of θ:



Then reflection in M_1 then reflection in M_2 is the same as rotating counterclockwise by 2θ.

So we know

$$\mathcal{Q} = -I_{|\psi_0\rangle} I_{|x_0\rangle}$$

is reflection in $\left|x_0^\perp\right\rangle$ then reflection in $\left|\psi_0^{\perp\perp}\right\rangle = |\psi_0\rangle$. So this is a rotation by 2α, where α is the angle between $\left|x_0^\perp\right\rangle$ and $|\psi_0\rangle$, i.e.

$$\sin\alpha = \cos\beta = \langle x_0|\psi_0\rangle\,.$$

To prove our second claim that $\mathcal{Q}$ acts as $-\mathbf{1}$ in $\mathcal{P}(x_0)^\perp$, we simply note that if $|\xi\rangle \in \mathcal{P}(x_0)^\perp$, then $|\xi\rangle \perp |\psi\rangle_0$ and $\xi \perp |x_0\rangle$. So both $I_{|x_0\rangle}$ and $I_{|\psi_0\rangle}$ fix $|\xi\rangle$.

In fact, Grover's algorithm is the best algorithm we can achieve.

Theorem. Let A be any quantum algorithm that solves the unique search problem with probability $1-\varepsilon$ (for any constant ε), with T queries. Then T is at least $O(\sqrt{N})$. In fact, we have

$$T \geq \frac{\pi}{4}(1-\varepsilon)\sqrt{N}.$$

So Grover's algorithm is not only optimal in the growth rate, but in the constant as well, asymptotically.

Proof is omitted.

Further generalizations

Suppose we have multiple good items instead, say r of them. We then replace I_{x_0} with I_f, where

$$I_f\,|x\rangle = \begin{cases} -\,|x\rangle & x \text{ good} \\ |x\rangle & x \text{ bad} \end{cases}$$

We run the same algorithm as before. We let

$$|\psi_{\text{good}}\rangle = \frac{1}{\sqrt{r}} \sum_{x \text{ good}} |x\rangle\,.$$

Then now $\mathcal{Q}$ is a rotation through 2α in the plane spanned by $|\psi_{\text{good}}\rangle$ and $|\psi_0\rangle$ with

$$\sin\alpha = \langle\psi_{\text{good}}|\psi_0\rangle = \sqrt{\frac{r}{N}}.$$

So for large N, we need

$$\frac{\pi/2}{2\sqrt{r/N}} = \frac{\pi}{4}\sqrt{\frac{N}{r}},$$

i.e. we have a $\sqrt{r}$ reduction over the unique case. We will prove that these numbers are right later when we prove a much more general result.

What if we don't know what r is? The above algorithm would not work, because we will not know when to stop the rotation. However, there are some tricks we can do to fix it. This involves cleverly picking angles of rotation at random, and we will not go into the details.

3.5 Amplitude amplification

In fact, the techniques from Grover's algorithm is completely general. Let G be any subspace ("good" subspace) of the state space $\mathcal{H}$, and $G^\perp$ be its orthogonal complement ("bad" subspace). Then

$$\mathcal{H} = G \oplus G^\perp.$$

Given any normalized vector $|\psi\rangle \in \mathcal{H}$, we have a unique decomposition with real, non-negative coefficients

$$|\psi\rangle = \sin\theta\,|\psi_g\rangle + \cos\theta\,|\psi_b\rangle$$

such that

$$|\psi_g\rangle \in G, \quad |\psi_b\rangle \in G^\perp$$

are normalized.

We define the reflections

$$I_{|\psi\rangle} = I - 2\,|\psi\rangle\langle\psi|, \quad I_g = I - 2P,$$

where P is the projection onto G given by

$$P = \sum_b |b\rangle\langle b|$$

for any orthonormal basis $\{|b\rangle\}$ of G. This P satisfies

$$P\,|\psi\rangle = \begin{cases} |\psi\rangle & |\psi\rangle \in G \\ 0 & |\psi\rangle \in G^\perp \end{cases}.$$

We now define the *Grover operator*

$$\mathcal{Q} = -I_\psi I_G.$$

Theorem (Amplitude amplification thoerem)**.** In the 2-dimensional subspace spanned by $|\psi_g\rangle$ and $|\psi\rangle$ (or equivalently by $|\psi_g\rangle$ and $|\psi_b\rangle$), where

$$|\psi\rangle = \sin\theta\,|\psi_g\rangle + \cos\theta\,|\psi_b\rangle,$$

we have that $\mathcal{Q}$ is rotation by 2θ.

Proof. We have

$$I_G |\psi_g\rangle = -|\psi_g\rangle, \quad I_G |\psi_b\rangle = |\psi_b\rangle.$$

So

$$\mathcal{Q} |\psi_g\rangle = I_\psi |\psi_g\rangle, \quad \mathcal{Q} |\psi_b\rangle = -I_\psi |\psi_b\rangle.$$

We know that

$$I_\psi = I - 2|\psi\rangle\langle\psi|.$$

So we have

$$\begin{aligned}
\mathcal{Q} |\psi_g\rangle &= I_\psi |\psi_g\rangle \\
&= |\psi_g\rangle - 2(\sin\theta |\psi_g\rangle + \cos\theta |\psi_b\rangle)(\sin\theta) \\
&= (1 - 2\sin^2\theta) |\psi_g\rangle - 2\sin\theta\cos\theta |\psi_b\rangle \\
&= \cos 2\theta |\psi_g\rangle - \sin 2\theta |\psi_b\rangle \\
\mathcal{Q} |\psi_b\rangle &= -I_\psi |\psi_b\rangle \\
&= -|\psi_b\rangle + 2(\sin\theta |\psi_g\rangle + \cos\theta |\psi_b\rangle)(\cos\theta) \\
&= 2\sin\theta\cos\theta |\psi_g\rangle + (2\cos^2\theta - 1) |\psi_b\rangle \\
&= \sin 2\theta |\psi_g\rangle + \cos 2\theta |\psi_b\rangle.
\end{aligned}$$

So this is rotation by 2θ. □

If we iterate this n times, then we have rotated by $2n\theta$, but we started at θ from the $|\psi_b\rangle$ direction. So we have

$$\mathcal{Q}^n |\psi\rangle = \sin(2n+1)\theta |\psi_g\rangle + \cos(2n+1)\theta |\psi_b\rangle.$$

If we measure $\mathcal{Q}^n |\psi\rangle$ for good versus bad, we know

$$\mathbb{P}(\text{good}) = \sin^2(2n+1)\theta,$$

and this is a maximum, when $(2n+1)\theta = \frac{\pi}{2}$, i.e.

$$n = \frac{\pi}{4\theta} - \frac{1}{2}.$$

For a general θ, we know that n is not a n integer. So we use n the nearest integer to $\frac{\pi}{4\theta} - \frac{1}{2}$, which is approximately

$$\frac{\pi}{4\theta} = O(\theta^{-1}) = O(1/\sin\theta) = O\left(\frac{1}{\|\text{good projection of } |\psi\rangle\|}\right).$$

Example. Suppose we want to do a Grover search for r good items in N objects. We start with

$$|\psi\rangle = \frac{1}{\sqrt{N}} \sum_{\text{all } x} |x\rangle = \sqrt{\frac{r}{N}} \left(\frac{1}{\sqrt{r}} \sum_{\text{good } x} |x\rangle\right) + \sqrt{\frac{N-r}{N}} \left(\frac{1}{\sqrt{N-r}} \sum_{\text{bad } x} |x\rangle\right).$$

Then G is the subspace spanned by the good x's, and

$$\sin\theta = \sqrt{\frac{r}{N}},$$

So $\mathcal{Q}$ is a rotation by 2θ, where

$$\theta = \sin^{-1}\sqrt{\frac{r}{N}} \approx \sqrt{\frac{r}{N}}$$

for $r \ll N$. So we will use $O(\sqrt{r/N})$ operations.

Example. Let A be any quantum circuit on start state $|0\cdots 0\rangle$. Then the final state is $A|0\cdots 0\rangle$. The good states are the desired computational outcomes. For example, if A is Shor's algorithm, then the desired outcomes might be the good c-values. We can write

$$A|0\cdots 0\rangle = a|\psi_g\rangle + b|\psi_b\rangle.$$

The probability of a success in one run is $|a|^2$. So we normally need $O(1/|a|^2)$ repetitions of A to succeed with a given constant probability $1-\varepsilon$.

Instead of just measuring the result and hoping for the best, we can use amplitude amplification. We assume we can check if x is good or bad, so we can implement I_G. We consider

$$|\psi\rangle = A|0\cdots 0\rangle.$$

Then we define

$$\mathcal{Q} = -I_{A|0\cdots 0\rangle} I_G = -AI_{|0\cdots 0\rangle}A^\dagger I_G.$$

Here we can construct $A^\dagger$ just by reversing the gates in A. So all parts are implementable.

By amplitude amplification, $\mathcal{Q}$ is rotation by 2θ, where $\sin\theta = |a|$. So after

$$n \approx \frac{\pi}{4\theta} = O(|a|^{-1})$$

repetitions, $A|0\cdots 0\rangle$ will be rotated to very near to $|\psi_g\rangle$, and this will succeed with high probability. This gives us a square root speedup over the naive method.

4 Measurement-based quantum computing

In this chapter, we are going to look at an alternative model of quantum computation. This is rather weird. Instead of using unitary gates and letting them act on state, we prepare a generic starting state known as a *graph state*, then we just keep measuring them. Somehow, by cleverly planning the measurements, we will be able to simulate any quantum computation in the usual sense with such things.

We will need a bunch of new notation.

Notation. We write

$$|\pm_\alpha\rangle = \frac{1}{\sqrt{2}}(|0\rangle \pm e^{-i\alpha}|1\rangle).$$

In particular, we have

$$|\pm_0\rangle = |\pm\rangle = \frac{1}{\sqrt{2}}(|0\rangle \pm |1\rangle)$$

Then

$$\mathcal{B}(\alpha) = \{|+_\alpha\rangle, |-_\alpha\rangle\}$$

is an orthonormal basis. We have 1-qubit gates

$$\mathsf{J}(\alpha) = \frac{1}{\sqrt{2}}\begin{pmatrix}1 & e^{i\alpha}\\ 1 & -e^{i\alpha}\end{pmatrix} = \mathsf{HP}(\alpha),$$

where

$$\mathsf{H} = \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}, \quad \mathsf{P}(\alpha) = \begin{pmatrix}1 & 0\\ 0 & e^{i\alpha}\end{pmatrix}.$$

We also have the "Pauli gates"

$$\mathsf{X} = \begin{pmatrix}0 & 1\\ 1 & 0\end{pmatrix}, \quad \mathsf{Z} = \begin{pmatrix}1 & 0\\ 0 & -1\end{pmatrix} = \mathsf{P}(\pi)$$

We also have the 2-qubit gates

$$\mathsf{E} = \mathsf{CZ} = \mathrm{diag}(1, 1, 1, -1).$$

We also have 1-qubit measurements

$$\mathsf{M}_i(\alpha) = \text{measurement of qubit } i \text{ in basis } \mathcal{B}(\alpha).$$

The outcome $|+_\alpha\rangle$ is denoted 0 and the outcome $|-_\alpha\rangle$ is denoted 1.

We also have $\mathsf{M}_i(\mathsf{Z})$, which is measurement of qubit i in the standard basis $\{|0\rangle, |1\rangle\}$.

Finally, we have the notion of a graph state. Suppose we have an undirected graph $G = (V, E)$ with vertices V and edges E with no self-loops and at most one edge between two vertices, we can define the *graph state* $|\psi_G\rangle$ that is a state of $|V|$ qubits as follows: for each vertex $i \in V$, introduce a qubit $|+\rangle_i$. For each edge $e : i \to j$, we apply E_{ij} (i.e. E operating on the qubits i and j). Since all these E_{ij} commute, the order does not matter.

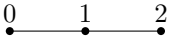
Example. If G_1 is

then we have

$$|\psi_{G_1}\rangle = E_{12}\,|+\rangle_1\,|+\rangle_2 = \frac{1}{2}[|00\rangle + |01\rangle + |10\rangle - |11\rangle],$$

and this is an entangled state.

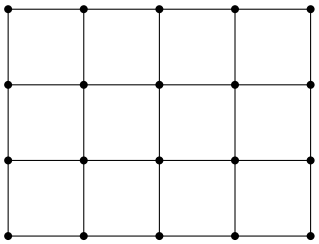
If G_2 is



then we have

$$|\psi_{G_2}\rangle = E_{12}E_{23}\,|+\rangle_1\,|+\rangle_2\,|+\rangle_3\,.$$

A *cluster state* is a graph state $|\psi_G\rangle$ for G being a rectangular $2D$ grid.



The main result of measurement-based quantum computation is the following:

Theorem. Let C be any quantum circuit on n qubits with a sequence of gates $U_1, \cdots, U_K$ (in order). We have an input state $|\psi_{\text{in}}\rangle$, and we perform Z-measurements on the output states on specified qubits $j = i_1, \cdots, i_k$ to obtain a k-bit string.

We can always simulate the process as follows:

(i) The starting resource is a graph state $|\psi_G\rangle$, where G is chosen depending on the connectivity structure of C.

(ii) The computational steps are 1-qubit measurements of the form $\mathsf{M}_i(\alpha)$, i.e. measurement in the basis $\mathcal{B}(\alpha)$. This is adaptive — α may depend on the (random) outcomes $s_1, s_2, \cdots$ of previous measurements.

(iii) The computational process is a prescribed (adaptive) sequence $\mathsf{M}_{i_1}(\alpha_1)$, $\mathsf{M}_{i_2}(\alpha_2)$, $\cdots$, $\mathsf{M}_{i_N}(\alpha_N)$, where the qubit labels $i_1, i_2, \cdots, i_N$ all distinct.

(iv) To obtain the output of the process, we perform further measurements $\mathsf{M}(\mathsf{Z})$ on k specified qubits not previously measured, and we get results $s_{i_1}, \cdots, s_{i_k}$, and finally the output is obtained by further (simple) *classical* computations on $s_{i_1}, \cdots, s_{i_k}$ as well as the previous $\mathsf{M}_i(\alpha)$ outcomes.

The idea of the last part is that the final measurement $s_{i_1}, \cdots, s_{i_k}$ has to be re-interpret in light of the results $M_i(\alpha_i)$.

This is a funny process, because the result of each measurement $M_i(\alpha)$ is uniformly random, with probability $\frac{1}{2}$ for each outcome, but somehow we can obtain useful information by doing adaptive measurements.

We now start the process of constructing such a system. We start with the following result:

Fact. The 1-qubit gates $\mathsf{J}(\alpha)$ with $\mathsf{E}_{i,i\pm1}$ is a universal set of gate.

In particular, any 1-qubit U is a product of 3 J's.

We call these $E_{i,i\pm1}$ *nearest neighbour* E_{ij}*'s*.

Proof. This is just some boring algebra. □

So we can assume that our circuit C's gates are all of the form $\mathsf{J}(\alpha)$'s or $\mathsf{E}'_{ij}s$, and it suffices to try to implement these gates in our weird system.

The next result we need is what we call the J-lemma:

Lemma (J-lemma)**.** Given any 1-qubit state $|\psi\rangle$, consider the state

$$\mathsf{E}_{12}(|\psi\rangle_1 |+\rangle_2).$$

Suppose we now measure $\mathsf{M}_1(\alpha)$, and suppose the outcome is $s_1 \in \{0, 1\}$. Then after measurement, the state of 2 is

$$\mathsf{X}^{s_1}\mathsf{J}(\alpha)|\psi\rangle.$$

Also, two outcomes $s = 0, 1$ always occurs with probability $\frac{1}{2}$, regardless of the values of $|\psi\rangle\, b, \alpha$.

Proof. We just write it out. We write

$$|\psi\rangle = a|0\rangle + b|1\rangle.$$

Then we have

$$\begin{aligned}\mathsf{E}_{12}(|\psi\rangle_1 |+\rangle_2) &= \frac{1}{\sqrt{2}}\mathsf{E}_{12}(a|0\rangle|0\rangle + a|0\rangle|1\rangle + b|1\rangle|0\rangle + b|1\rangle|1\rangle)\\ &= \frac{1}{\sqrt{2}}(a|0\rangle|0\rangle + a|0\rangle|1\rangle + b|1\rangle|0\rangle - b|1\rangle|1\rangle)\end{aligned}$$

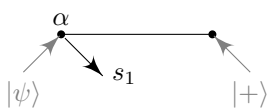
So if we measured 0, then we would get something proportional to

$$\begin{aligned}\langle +_\alpha|_1 \mathsf{E}_{12}(|\psi\rangle_1 |+\rangle_2) &= \frac{1}{2}(a|0\rangle + a|1\rangle + be^{i\alpha}|0\rangle - be^{i\alpha}|1\rangle)\\ &= \frac{1}{2}\begin{pmatrix}1 & e^{i\alpha}\\ 1 & -e^{i\alpha}\end{pmatrix}\begin{pmatrix}a\\ b\end{pmatrix},\end{aligned}$$

as required. Similarly, if we measured 1, then we get $\mathsf{XJ}(\alpha)|\psi\rangle$. □

We will usually denote processes by diagrams. In this case, we started with the graph state

and the measurement can be pictured as



If we measure Z, we denote that by

In fact, this can be extended if 1 is just a single qubit part of a larger multi-qubit system $1\mathcal{S}$, i.e.

Lemma. Suppose we start with a state

$$|\psi\rangle_{1\mathcal{S}} = |0\rangle_1 |a\rangle_{\mathcal{S}} + |1\rangle_1 |b\rangle_{\mathcal{S}}.$$

We then apply the J-lemma process by adding a new qubit $|+\rangle$ for $2 \notin \mathcal{S}$, and then query 1. Then the resulting state is

$$\mathsf{X}_2^{s_1} \mathsf{J}_2(\alpha) |\psi\rangle_{2\mathcal{S}}.$$

So the J-lemma allows us to simulate J-gates with measurements. But we want to do many J gates. So we need the concatenation lemma:

Lemma (Concatenation lemma)**.** If we concatenate the process of J-lemma on a row of qubits $1, 2, 3, \cdots$ to apply a sequence of $\mathsf{J}(\alpha)$ gates, then all the entangling operators $\mathsf{E}_{12}, \mathsf{E}_{23}, \cdots$ can be done *first* before any measurements are applied.

It is a fact that for any composite quantum system $A \otimes B$, any local actions (unitary gates or measurements) done on A always commutes with anything done on B, which is easy to check by expanding out the definition. So the proof of this is simple:

Proof. For a state $|\psi\rangle_1 |+\rangle_2 |+\rangle_3 \cdots$, we can look at the sequence of J-processes in the sequence of operations (left to right):

$$\mathsf{E}_{12}\mathsf{M}_1(\alpha_1)\mathsf{E}_{23}\mathsf{M}_2(\alpha_2)\mathsf{E}_{34}\mathsf{M}_3(\alpha_3)\cdots$$

It is then clear that each E_{ij} commutes with all the measurements before it. So we are safe. □

We can now determine the measurement-based quantum computation process corresponding to a quantum circuit C of gates $U_1, U_2, \cdots, U_K$ with each U_i either a $\mathsf{J}(\alpha)$ or a nearest-neighbour E_{ij}. We may wlog assume the input state to C is

$$|+\rangle \cdots |+\rangle$$

as any 1-qubit product state may be written as

$$|\psi\rangle = U |+\rangle$$

for suitable U, which is then represented as at most three $\mathsf{J}(\alpha)$'s. So we simply prefix C with these $J(\alpha)$ gates.

Example. We can write $|j\rangle$ for $j = 0, 1$ as

$$|j\rangle = \mathsf{X}^j \mathsf{H} |+\rangle.$$

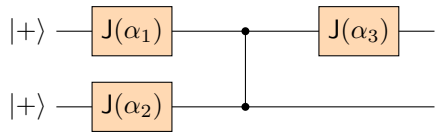
We also have

$$\mathsf{H} = \mathsf{J}(0), \quad \mathsf{X} = \mathsf{J}(\pi)\mathsf{J}(0).$$

So the idea is that we implement these $\mathsf{J}(\alpha)$ gates by the J-processes we just described, and the nearest-neighbour E_{ij} gates will just be performed when we create the graph state.

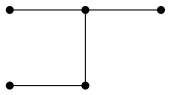
We first do a simple example:

Example. Consider the circuit C given by



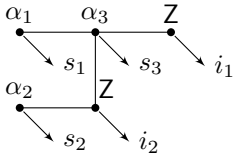
where the vertical line denotes the E_{12} operators. At the end, we measure the outputs i_1, i_2 by $\mathsf{M}(Z)$ measurements.

We use the graph state

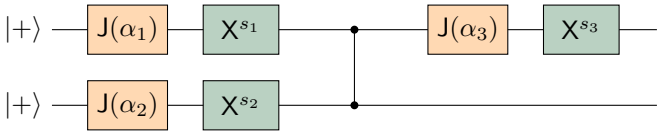


In other words, we put a node for a $|+\rangle$, horizontal line for a $\mathsf{J}(\alpha)$ and a vertical line for an E.

If we just measured all the qubits for the J-process in the order $\alpha_1, \alpha_2, \alpha_3$, and then finally read off the final results i_1, i_2:



then we would have effected the circuit



Now the problem is to get rid of the X^i's. We know each X^i comes with probability $\frac{1}{2}$. So the probability of them all not appearing is tiny for more complicated circuits, and we cannot just rely on pure chance for it to turn out right.

To deal with the unwanted X^i "errors", we want to commute them out to the end of the circuit. But they do not commute, so we are going to use the following commutation relations:

$$J(\alpha)X = e^{i\alpha}ZJ(-\alpha)$$

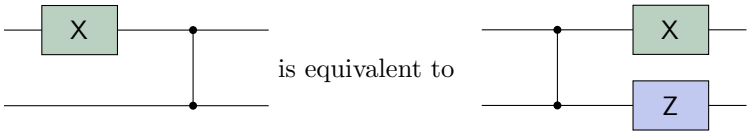
In other words, up to an overall phase, the following diagrams are equivalent:

More generally, we can write

$$\begin{aligned} \mathsf{J}_i(\alpha)\mathsf{X}_i^s &= e^{-i\alpha s}\mathsf{Z}_i^s\mathsf{J}_i((-1)^s\alpha) \\ \mathsf{J}_i(\alpha)\mathsf{Z}_i^s &= \mathsf{X}_i^s\mathsf{J}_i(\alpha) \\ \mathsf{E}_{ij}\mathsf{Z}_i^s &= \mathsf{Z}_i^s\mathsf{E}_{ij} \\ \mathsf{E}_{ij}\mathsf{X}_i^s &= \mathsf{X}_i^s\mathsf{Z}_i^s\mathsf{E}_{ij} \end{aligned}$$

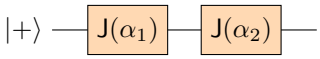
Here the subscripts tell us which qubit the gates are acting on.

The last one corresponds to



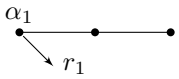
All of these are good, except for the first one where we have a funny phase and the angle is negatived. The phase change is irrelevant because it doesn't affect measurements, but the sign changes are problematic. To fix this, we need to use adaptive measurements.

Example. Consider the simpler 1-qubit circuit

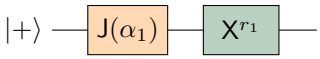


We first prepare the graph sate

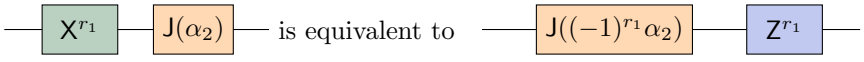
We now measure the first qubit to get



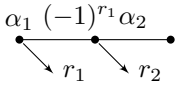
We have thus done



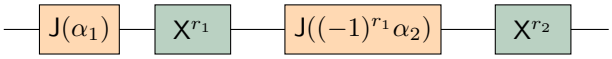
To deal with the unwanted X^{r_1}, we note that



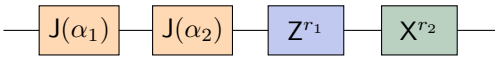
So we adapt the sign of the second measurement angle to depend on the previous measurement result:



Then this measurement results in



which is equivalent to



If we had further J-gates, we need to commute both Z^{r_1} and X^{r_2} over.

Note that while we are introducing a lot of funny X's and Z's, these are all we've got, and the order of applying them does not matter, as they anti-commute:

$$\mathsf{XZ} = -\mathsf{ZX}.$$

So if we don't care about the phase, they effectively commute.

Also, since $\mathsf{X}^2 = \mathsf{Z}^2 = I$, we only need to count the number of X's and Z's mod 2, which is very helpful.

Now what do we do with the Z and X at the end? For the final Z-measurement, having moved everything to the end, we simply reinterpret the final, actual Z-measurement result j:

(i) The Z-gate does not affect outcome or probability of a Z-measurement, becasuse if
$$|\psi\rangle = a\,|0\rangle + b\,|1\rangle\,,$$
then
$$\mathsf{Z}\,|\psi\rangle = a\,|0\rangle - b\,|1\rangle\,.$$
So the probabilities of $|0\rangle$ and $|1\rangle$ are $|a|^2$ and $|b|^2$ regardless.

(ii) The X gate simply interchanges the labels, while leavining probabilities the same, because if
$$|\psi\rangle = a\,|0\rangle + b\,|1\rangle\,,$$
then
$$\mathsf{X}\,|\psi\rangle = a\,|1\rangle + b\,|0\rangle\,.$$

So we ignore all Z-errors, and for each X^r error, we just modify the seen measurement outcome j by $j \mapsto j \oplus r$.

If we actually implement measurement-based quantum computations, the measurements can always be done "left to right", implementing the gates in order. However, we don't have to do that. Recall that quantum operations on disjoint qubits always commute. Since the only thing we are doing are measurements, *all* $M_i(\alpha)$ measurements can be performed simultaneously if the angles α do not depend on other measurements. This gives us a novel way to parallel a computation.

For example, in our simple example, we can start by first measuring r_1 and j, and then measuring r_2 after we know r_1. In particular, we can *first* measure the "answer" j, before we do any other thing! The remaining measurements just tell us how we should interpret the answer.

In general, we can divide the measurements into "layers" — the first layer consists of all measurements that do not require any adaptation. The second layer then consists of the measurements that only depends on the first layer. The *logical depth* is the least number of layers needed, and this somewhat measures the complexity of our circuit.

5 Phase estimation algorithm

We now describe a quantum algorithm that estimates the eigenvalues of a unitary operator. Suppose we are given a unitary operator U and an eigenstate $|v_\varphi\rangle$. Then we can write

$$U|V_\varphi\rangle = e^{2\pi i\varphi}|v_\varphi\rangle$$

with $0 \le \varphi < 1$. Our objective is to estimate φ to n binary bits of precision:

$$\varphi \approx 0.i_1i_2i_3\cdots i_n = \frac{i_1}{2} + \frac{i_2}{2^2} + \cdots + \frac{i_n}{2^n}.$$

We will need the controlled U^k gate $\mathsf{c}\text{-}U^k$ for integers k, defined by

$$\begin{aligned}\mathsf{c}\text{-}U^k|0\rangle|\xi\rangle &= |0\rangle|\xi\rangle\\ \mathsf{c}\text{-}U^k|1\rangle|\xi\rangle &= |1\rangle U^k|\xi\rangle,\end{aligned}$$

where $|0\rangle, |1\rangle$ are 1-qubit states, and $|\xi\rangle$ is a general d-dimensional register.

Note that we have

$$U^k|v_\varphi\rangle = e^{2\pi ik\varphi}|v_\varphi\rangle,$$

and we have

$$\mathsf{c}\text{-}U^k = (\mathsf{c}\text{-}U)^k,$$

Note that if we are given U as a formula or a circuit *description*, then we can readily implement $\mathsf{c}\text{-}U$ by adding control to each gate. However, if U is a quantum black-box, then we need further information. For example, it suffices to have an eigenstate $|\alpha\rangle$ with *known* eigenvalue $e^{i\alpha}$. However, we will not bother ourselves with that, and just assume that we can indeed implement it.

In fact, we will use a "generalized" controlled U given by

$$|x\rangle|\xi\rangle \mapsto |x\rangle U^x|\xi\rangle,$$

where $|x\rangle$ has n qubits. We will make this from $\mathsf{c}\text{-}U^k = (\mathsf{c}\text{-}U)^k$ as follows: for

$$x = x_{n-1}\cdots x_1x_0 = x_0 + 2^1x_1 + 2^2x_2 + \cdots + 2^{n-1}x_{n-1},$$

we write $\mathsf{c}\text{-}U_i^k$ for the controlled U^k controlled by i. Then we just construct

$$U_0^{2^0}U_1^{2^1}\cdots U_{n-1}^{2^{n-1}}.$$

Now if input $|\xi\rangle = |v_\varphi\rangle$, then we get

$$e^{2\pi i\varphi x}|x\rangle|v_\varphi\rangle.$$

To do phase estimation, we superpose the above over all $x = 0, 1, 2, \cdots, 2^{n-1}$ and use $|\xi\rangle = |v_\varphi\rangle$. So we construct our starting state by

$$|s\rangle = \mathsf{H}\otimes\cdots\otimes\mathsf{H}|0\cdots0\rangle = \frac{1}{\sqrt{2^n}}\sum_{\text{all } x}|x\rangle.$$

Now if we apply the generalized control U, we obtain

$$\underbrace{\left(\frac{1}{\sqrt{2^n}}\sum_x e^{2\pi i\varphi x}|x\rangle\right)}_{|A\rangle}|v_\psi\rangle.$$

Finally, we apply the inverse Fourier transform $\mathsf{QFT}_{2^n}^{-1}$ to $|A\rangle$ and measure to see $y_0, y_1, \cdots, y_{n-1}$ on lines $0, 1, \cdots, n-1$. Then we simply output

$$0.y_0y_1\cdots y_{n-1} = \frac{y_0}{2} + \frac{y_1}{4} + \cdots + \frac{y_{n-1}}{2^n} = \frac{y_0y_1\cdots y_{n-1}}{2^n}.$$

as the estimate of φ.

Why does this work? Suppose φ actually only had n binary digits. Then we have

$$\varphi = 0.z_0z_1\cdots\cdots z_{n-1} = \frac{z}{2^n},$$

where $z \in \mathbb{Z}_{2^n}$. Then we have

$$|A\rangle = \frac{1}{\sqrt{2^n}} \sum_x 2^{2\pi i x z/2^n} |x\rangle,$$

which is the Fourier transform of $|z\rangle$. So the inverse Fourier transform of $|A\rangle$ is exactly $|Z\rangle$ and we get φ exactly with certainty.

If φ has more than n bits, say

$$\varphi = 0.z_0z_1\cdots z_{n-1}z_nz_{n+1}\cdots,$$

then we have

Theorem. If the measurements in the above algorithm give $y_0, y_1, \cdots, y_n$ and we output

$$\theta = 0.y_0y_1\cdots y_{n-1},$$

then

(i) The probability that θ is φ to n digits is at least $\frac{4}{\pi^2}$.

(ii) The probability that $|\theta - \varphi| \geq \varepsilon$ is at most $O(1/(2^n\varepsilon))$.

The proofs are rather boring and easy algebra.

So for any fixed desired accuracy ε, the probability to fail to get φ to this accuracy falls *exponentially* with n.

Note that if $\mathsf{c}\text{-}U^{2^k}$ is implemented as $(\mathsf{c}\text{-}U)^{2^k}$, then the algorithm would need

$$1 + 2 + 4 + \cdots + 2^{n-1} = 2^{n-1}$$

many $\mathsf{c}\text{-}U$ gates. But for some special U's, this $\mathsf{c}\text{-}U^{2^k}$ can be implemented in polynomial time in k.

For example, in *Kitaev's factoring algorithm*, for $\mathrm{hcf}(a, N) = 1$, we will use the function

$$U : |m\rangle \mapsto |am \bmod N\rangle.$$

Then we have

$$U^{2^k} |m\rangle = \left|a^{2^k} m\right\rangle,$$

which we can implement by repeated squaring.

Now what if we didn't have an eigenstate to being with? If instead of $|v_\varphi\rangle$, we used a general input state $|\xi\rangle$, then we can write

$$|\xi\rangle = \sum_j c_j \left|v_{\varphi_j}\right\rangle,$$

where

$$U\left|v_{\varphi_j}\right\rangle = e^{2\pi i\varphi_j}\left|v_{\varphi_j}\right\rangle.$$

Then in the phase estimation algorithm, just before the final measurement, we have managed to get ourselves

$$|0\cdots 0\rangle\,|\xi\rangle \to \sum_j c_j\,|\varphi_j\rangle\left|v_{\varphi_j}\right\rangle.$$

Then when we measure, we get *one* of the φ_j's (or an approximation of it) with probability $|c_j|^2$. Note that this is *not* some average of them. Of course, we don't know which one we got, but we still get some meaningful answer.

Quantum counting

An application of this is the quantum counting problem. Given $f : B_n \to B$ with k good x's, we want to estimate the number k.

Recall the Grove iteration operator $\mathcal{Q}_G$ is rotation through 2θ in a 2-dimensional plane spanned by

$$|\psi_0\rangle = \frac{1}{\sqrt{2^n}}\sum_x |x\rangle$$

and its good projection, and θ is given by

$$\sin\theta \approx \theta = \sqrt{\frac{k}{N}}.$$

Now the eigenvalues of this rotation in the plane are

$$e^{2i\theta}, e^{-2i\theta}.$$

So either eigenvalue will suffice to get k.

We will equivalently write

$$e^{i2\theta} = e^{2\pi i\varphi}$$

with

$$0 \le \varphi < 1.$$

Then $\pm 2\theta$ is equivalent to φ or $1-\varphi$, where φ is small.

Now we don't have an eigenstate, but we can start with any state in the plane, $|\psi_0\rangle$. We then do phase estimation with it. We will then get either φ or $1-\varphi$ with some probabilities, but we don't mind which one we get, since we can obtain one from the other, and we can tell them apart because φ is small.

6 Hamiltonian simulation

So. Suppose we did manage to invent a usable quantum computer. What would it be good for? Grover's algorithm is nice, but it seems a bit theoretical. You might say we can use Shor's algorithm to crack encryption, but then if quantum computers are available, then no one would be foolish enough to use encryption that is susceptible to such attacks. So what can we actually do?

One useful thing would be to simulate physical systems. If we have a quantum system and with n qubits, then we want to simulate its evolution over time, as governed by Schrödinger's equation. Classically, a n-qubit system is specified by 2^n complex numbers, so we would expect any such algorithm to have performance at best $O(2^n)$. However, one would imagine that to simulate a quantum n-qubit system in a quantum computer, we only need n-qubits! Indeed, we will see that we will be able to simulate quantum systems in polynomial time in n.

In a quantum physical system, the state of the system is given by a state $|\psi\rangle$, and the evolution is governed by a *Hamiltonian* H. This is a self-adjoint (Hermitian) operator, and in physics, this represents the energy of the system. Thus, we have

$$\langle\psi| H |\psi\rangle = \text{average value obtained in measurement of energy.}$$

The time evolution of the particle is given by the Schrödinger equation

$$\frac{\mathrm{d}}{\mathrm{d}t} |\psi(t)\rangle = -iH |\psi(t)\rangle .$$

We'll consider only time-independent Hamiltonians $H(t) = H$. Then the solution can be written down as

$$|\psi(t)\rangle = e^{-iHt} |\psi(0)\rangle .$$

Here e^{-iHt} is the matrix exponential given by

$$e^A = I + A + \frac{A^2}{2} + \cdots ,$$

Thus, given a Hamiltonian H and a time t, we want to simulate $U(t) = e^{-iHt}$ to suitable approximations.

Before we begin, we note the following useful definitions:

Definition (Operator norm)**.** The *operator norm* of an operator A is

$$\|A\| = \max_{\||\psi\rangle\|=1} \|A |\psi\rangle\| .$$

If A is diagonalizable, then this is the maximum eigenvalue of A.

The following properties are easy to see:

Proposition.

$$\|A + B\| \leq \|A\| + \|B\|$$
$$\|AB\| \leq \|A\| \, \|B\| .$$

We now begin. There will be a slight catch in what we do. We will have to work with special Hamiltonians known as *k-local Hamiltonians* for a fixed k. Then for this fixed k, the time required to simulate the system will be polynomial in n. However, we should not expect the complexity to grow nicely as we increase k!

So what is a k-local Hamiltonian? This is a Hamiltonian in which each interaction governed by the Hamiltonian only involves k qubits. In other words, this Hamiltonian can be written as a sum of operators, each of which only touches k qubits. This is not too bad a restriction, because in real life, most Hamiltonians are indeed local, so that if each qubit represents a particle, then the behaviour of the particle will only be affected by the particles near it.

Definition (k-local Hamiltonian)**.** We say a Hamiltonian H is k-local (for k a fixed constant) on n qubits if

$$H = \sum_{j=1}^{m} H_j,$$

where each H_j acts on at most k qubits (not necessarily adjacent), i.e. we can write

$$H_j = \tilde{H}_j \otimes I,$$

where $\tilde{H}_j$ acts on some k qubits, and I acts on all other qubits as the identity.

The number m of terms we need is bounded by

$$m \leq \binom{n}{k} = O(n^k),$$

which is polynomial in n.

Example. The Hamiltonian

$$H = \mathsf{X} \otimes I \otimes I - \mathsf{Z} \otimes I \otimes Y$$

is 2-local on 3 qubits.

We write $M_{(i)}$ to denote the operator M acting on the ith qubit.

Example. We could write

$$X \otimes I \otimes I = X_{(1)}.$$

Example (Ising model)**.** The *Ising model* on an $n \times n$ square lattice of qubits is given by

$$H = \mathsf{J} \sum_{i,j=1}^{n-1} \mathsf{Z}_{(i,j)}\mathsf{Z}_{i,j+1} + \mathsf{Z}_{(i,j)}\mathsf{Z}_{(i+1,j)}.$$

Example (Heisenberg model)**.** The Heisenberg model on a line is given by

$$H = \sum_{i=1}^{n-1} J_x \mathsf{X}_{(i)}\mathsf{X}_{(i+1)} + J_y \mathsf{Y}_{(i)}\mathsf{Y}_{(i+1)} + J_z \mathsf{Z}_{(i)}\mathsf{Z}_{(i+1)},$$

where J_x, J_y and J_z are real constants.

This is useful in modelling magnetic system.

The idea is that we simulate each $e^{iH_j t}$ separately, and then put them together. However, if $\{H_j\}$ doesn't commute, then in general

$$e^{-i\sum_j H_j t} \neq \prod_j e^{-iH_j t}.$$

So we need to somehow solve this problem. But putting it aside, we can start working on the quantum simulation problem.

We will make use of the following theorem:

Theorem (Solovay-Kitaev theorem)**.** Let U be a unitary operator on k qubits and S any universal set of quantum gates. Then U can be approximated to within ε using $O(\log^c \frac{1}{\varepsilon})$ from S, where $c < 4$.

In other words, we can simulate each $e^{-iH_j t}$ with very modest overhead in circuit size for improved error, *assuming we fix k.*

Proof. Omitted. □

We will also need to keep track of the accumulation of errors. The following lemma will be useful:

Lemma. Let $\{U_i\}$ and $\{V_i\}$ be sets of unitary operators with

$$\|U_i - V_i\| \leq \varepsilon.$$

Then

$$\|U_m \cdots U_1 - V_m \cdots V_1\| \leq m\varepsilon.$$

This is remarkable!

Proof. See example sheet 2. The idea is that unitary gates preserve the size of vectors, hence do not blow up errors. □

We start by doing a warm-up: we solve the easy case where the terms in the Hamiltonian commute.

Proposition. Let

$$H = \sum_{j=1}^{m} H_j$$

be any k-local Hamiltonian with commuting terms.

Then for any t, e^{-iHt} can be approximated to within ε by a circuit of

$$O\left(m \operatorname{poly}\left(\log\left(\frac{m}{\varepsilon}\right)\right)\right)$$

gates from any given universal set.

Proof. We pick $\varepsilon' = \frac{\varepsilon}{m}$, and approximate $e^{-iH_j t}$ to within ε'. Then the total error is bounded by $m\varepsilon' = \varepsilon$, and this uses

$$O\left(m \operatorname{poly}\left(\log\left(\frac{m}{\varepsilon}\right)\right)\right)$$

gates. □

We now do the full non-commutative case. To do so, we need to keep track of how much $e^{iH_i t}e^{-iH_j t}$ differs from $e^{i(H_i+H_j)t}$.

Notation. For a matrix X, we write

$$X + O(\varepsilon)$$

for $X + E$ with $\|E\| = O(\varepsilon)$.

Then we have

Lemma (Lie-Trotter product formula)**.** Let A, B be matrices with $\|A\|, \|B\| \leq K < 1$. Then we have

$$e^{-iA}e^{-iB} = e^{-i(A+B)} + O(K^2).$$

Proof. We have

$$\begin{aligned} e^{-iA} &= 1 - iA + \sum_{k=2}^{\infty} \frac{(iA)^k}{k!} \\ &= I - iA + (iA)^2 \sum_{k=0}^{\infty} \frac{(-iA)^k}{(k+2)!} \end{aligned}$$

We notice that $\|(iA)^2\| \leq K^2$, the final sum has norm bounded by $e^K < e$. So we have

$$e^{-iA} = I - iA + O(K^2).$$

Then we have

$$\begin{aligned} e^{-iA}e^{-iB} &= (I - iA + O(K^2))(I - iB + O(K^2)) \\ &= I - i(A + B) + O(K^2) \\ &= e^{-i(A+B)} + O(K^2). \end{aligned}$$

Here we needed the fact that $\|A + B\| \leq 2K = O(K)$ and $\|AB\| \leq K^2 = O(K^2)$. □

We now apply this repeatedly to accumulate sums $H_1, H_2, .., H_m$ in the exponent. First of all, we note that if each $\|H_i\| < K$, then $\|H_i + \cdots + H_\ell\| < \ell K$. We want this to be < 1 for all $\ell \leq m$. So for now, we assume $K < \frac{1}{m}$. Also, we take $t = 1$ for now. Then consider

$$\begin{aligned} e^{-iH_1}e^{-iH_2}\cdots e^{-iH_m} &= (e^{-i(H_1+H_2)} + O(K^2))e^{-iH_3}\cdots e^{-iH_m} \\ &= e^{-i(H_1+H_2)}e^{-iH_3}\cdots e^{-iH_m} + O(K^2) \\ &= e^{-i(H_1+H_2+H_3)}e^{-iH_4}\cdots e^{-iH_m} + O((2K)^2) + O(K^2) \\ &= e^{-i\sum H_j} + O(m^3K^2), \end{aligned}$$

where we used the fact that

$$1^2 + 2^2 + \cdots + m^2 = O(m^3).$$

We write the error as Cm^3K^2.

This is fine if K is super small, but it won't be in general. For general K and t values, we introduce a large N such that

$$\left\| \frac{H_j t}{N} \right\| < \frac{Kt}{N} \leq \tilde{K} < 1.$$

In other words, we divide time up into small $\frac{t}{N}$ intervals. We then try to simulate

$$U = e^{-i(H_1 + \cdots + H_m)t} = \left(e^{-i\left(\frac{H_1 t}{N} + \ldots + \frac{H_n t}{N}\right)} \right)^N.$$

This equality holds because we know that $\frac{t}{N}(H_1 + \cdots + H_n)$ commutes with itself (as does everything in the world).

We now want to make sure the final error for U is $< \varepsilon$. So we know each term $e^{-i\left(\frac{H_1 t}{n} + \ldots + \frac{H_n t}{n}\right)}$ needs to be approximated to $\frac{\varepsilon}{N}$. So using our previous formula, we want that

$$Cm^3 \tilde{K}^2 < \frac{\varepsilon}{N},$$

Doing some algebraic manipulation, we find that we need

$$N > \frac{Cm^3 K^2 t^2}{\varepsilon}.$$

We now have Nm gates of the form $e^{iH_j t/N}$. So the circuit size is at most

$$O\left(\frac{m^4 (Kt)^2}{\varepsilon} \right).$$

Recall for n-qubits, a general k-local Hamiltonian has $m = O(n^k)$. So the circuit size is

$$|\mathcal{C}| = O\left(\frac{n^{4k} (Kt)^2}{\varepsilon} \right).$$

Now this is in terms of the number of $e^{iH_j t/N}$ gates. If we want to express this in terms of universal gates, then each gate needs to be approximated to $O(\varepsilon/|\mathcal{C}|)$. We then need $O(\log^c(\frac{|\mathcal{C}|}{\varepsilon}))$ gates for each, for some $c < 4$. So we only get a modest extra multiplicative factor in $|\mathcal{C}|$.

Note that for a fixed n with a variable t, then a quantum process e^{-iHt} runs in time t, but our simulation needs time $O(t^2)$. This can be improved to $O(t^{1+\delta})$ for any $\delta > 0$ by using "better" Lie-Trotter expansions.

Local Hamiltonian ground state problem

There are many other things we might want to do with a k-local Hamiltonian. One question we might be interested in is the eigenvalues of H. Suppose we are given a 5-local Hamiltonian

$$H = \sum_{i=1}^{m} H_j$$

on n qubits (this was the original result proved). We suppose $\|H_i\| < 1$, and we are given two numbers $a < b$, e.g. $a = \frac{1}{3}$ and $b = \frac{2}{3}$. We are promised that the smallest eigenvalue E_0 of H is $< a$ or $> b$. The problem is to decide whether $E_0 < a$.

The reason we have these funny a, b is so that we don't have to worry about precision problems. If we only had a single a and we want to determine if $E_0 > a$ or $E_0 < a$, then it would be difficult to figure out if E_0 happens to be very close to a.

Kitaev's theorem says that the above problem is complete for a complexity class known as **QMA**, i.e. it is the "hardest" problem in **QMA**. In other words, any problem in **QMA** can be translated into a local Hamiltonian ground state problem with polynomial overhead. A brief survey can be found on arXiv:quant-ph/0210077.

What is this **QMA**? We will not go into details, but roughly, it is a quantum version of **NP**. In case you are wondering, MA stands for Merlin and Arthur...

Index

QFT_n, 16
J-lemma, 36
k-local Hamiltonian, 45
BPP, 7
P, 7
BQP, 10
NP, 26
QMA, 49

amplitude amplification theorem, 31
approximate universality, 10

balanced vs constant problem, 13
boolean satisfiability problem, 26
bounded error, probabilistic polynomial time, 7

circuit model, 8
complexity, 6
computational model, 6

decision problem, 5
Deutsch-Jozsa algorithm, 13

graph state, 34
Grover iteration operator, 28
Grover operator, 31
Grover's algorithm, 27, 28

Halting problem, 5
Hamiltonian, 44
  k-local, 45
Heisenberg model, 45
hidden subgroup problem, 19

input register, 11
input string, 5
Ising model, 45

Kitaev's factoring algorithm, 42

language, 5
Lie-Trotter product formula, 47
logical depth, 40

nearest neighbour E_{ij}'s, 36
non-deterministic computation, 26
non-deterministic polynomial time, 26

operator norm, 44
oracle, 13
output register, 11

Periodicity problem, 17
phase estimation, 41
phase kickback, 13
polynomial growth, 7
polynomial time, 7
polynomial time verifier, 26

quantum counting, 43
quantum Fourier transform, 16
query complexity, 13

SAT, 26
SAT problem, 15
satisfiability problem, 15
Simon's algorithm, 15
Simon's problem, 15
Solovay-Kitaev theorem, 46

Turing machine, 8

universal set, 8

verifier, 26